AD-A100 776 ROCKWELL INTERNATIONAL ANAHEIM CA F/G 5/8
PROGRAMMABLE IMAGE PROCESSING ELEMENT.(U)
FEB 81 S A WHITE F33615-79-C-1905
UNCLASSIFIED C80-702/501 AFWAL-TR-80-1208 NL

1 OF 1
AD
A100776

END
DATE
FILMED
7-81
DTIC

LEVEL II

2

AFWAL-TR-80-1208

PROGRAMMABLE IMAGE PROCESSING ELEMENT

ROCKWELL INTERNATIONAL
3370 MIRALOMA AVENUE
ANAHEIM, CA 92803

FEBRUARY, 1981

TECHNICAL REPORT AFWAL-TR-80-1208
Final Report for period September 1979 — September 1980

Approved for public release; distribution unlimited.

AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

NOTICE

This technical report has been reviewed and is approved for publication.

Guy D. Couturier

GUY D. COUTURIER
PROJECT ENGINEER

Robert E. Conklin

ROBERT E. CONKLIN,
CHIEF, PROCESSOR
TECHNOLOGY GROUP

FOR THE COMMANDER

Stanley E. Wagner

STANLEY E. WAGNER, CHIEF
MICROELECTRONICS BRANCH
ELECTRONIC TECHNOLOGY DIVISION

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFWAL-TR-80-1208	2. GOVT ACCESSION NO. AD-A100 776	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) PROGRAMMABLE IMAGE PROCESSING ELEMENT		5. TYPE OF REPORT & PERIOD COVERED Final Report, Sept. 1979 — Sept. 1980
		6. PERFORMING ORG. REPORT NUMBER C80-703/501
7. AUTHOR(s) Stanley A. White		8. CONTRACT OR GRANT NUMBER(s) F33615-79-C-1905
9. PERFORMING ORGANIZATION NAME AND ADDRESS Rockwell International 3370 Miraloma Avenue Anaheim, CA 92803		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 6096 30 07
11. CONTROLLING OFFICE NAME AND ADDRESS Avionics Laboratory (AFWAL/AADE) Air Force Wright Aeronautical Laboratories (AFSC) Wright-Patterson AFB, Ohio 45433		12. REPORT DATE February 1981
		13. NUMBER OF PAGES 59
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Parts of section 3 to be published, "An Architecture for a High-Speed Digital Signal Processing Device," IEEE International Symposium on Circuits and Systems, Chicago, Ill., April 1981. Parts of section 4 to be published, "An Architecture for a Digital Programmable Image Processing Element," IEEE Conference on Acoustics, Speech, and Signal Processing, Atlanta, GA, March 1981.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

vector multiplier	sliding window
inner product	transversal filter
Peled-Liu mechanization	multiply/accumulate
distributed arithmetic	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The generic signal-processing equations were analyzed and two specific architectures were evolved for the efficient implementation of these equations. The first architecture which was considered consisted of a set for four arithmetic sections. Each section was capable of combining a pair of 4-element vectors, 16 bits per element, using distributed arithmetic. The second architectural design was pursued which consisted of a single arithmetic section, to produce the inner product $y = \sum_{n=1}^{9} a_n x_n$ every 90 ns using a modified Peled-Liu algorithm.
(Over)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Input word lengths are limited to 8 bits, 2's complement. Coefficients are variable. The output is full precision. Input data (x's) may be loaded in parallel sequentially, or in parallel 3 at a time; or serially 1, 3 or 9 at a time. The device can be programmed to function as a sliding window, a transversal filter, or a vector multiplier. Multiple devices may be chained together to increase computational accuracy or to extend filter lengths beyond 9 taps.

PREFACE

The Programmable Image Processing Element (PIPE) program was performed by Rockwell International, 3370 Miraloma Avenue, Anaheim, California 92803, under Contract F33615-79-C-1905. The results were first published in the final report draft in October 1980 bearing Rockwell's document number C80-703/501.

The program was conducted from September 1979 through September 1980. The program monitors were Dr. Ron Belt and Mr. Guy Couturier, Avionics Laboratory (AFWAL/AADE), Air Force Wright Aeronautical Laboratories (AFSC), Wright Patterson Air Force Base, Ohio 45433.

This document is unclassified; no part of it has been taken from a classified document.

Accession For	
NTIS GRA&I	✓
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A	

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. Introduction

The Programmable Image Processing Element (PIPE) program, which was conducted by Rockwell International from September 1979 through September 1980, consisted of three actual phases:

1. The definition of a programmable device which could form the inner product of a pair of 16-element vectors;
2. the definition of a programmable device which could form the inner product of a pair of 9-element vectors;
3. the partial logical design, simulation, and layout of the device defined in 2.

Although a proposal was submitted to carry the program through the making and delivery of optomasks, that proposal was not accepted due to lack of funds.

This report contains four chapters. Chapter 2 addresses the rationale for the existence of the PIPE, Chapter 3 describes the device which was defined to combine a pair of 16-element vectors, and Chapter 4 describes the device which was defined to combine a pair of 9-element vectors.

The summary of the device design effort is contained in the Appendix.

2. Why PIPE?

2.1 Signal Processing Requirements of DoD

Within any digital communications system, or radar, or sonar, or control system we encounter innumerable requirements which are in a form known as a digital filter. To the signal this function is analagous to the familiar analog filters such as bandpass or low-pass or high-pass. These filters are described by equations of the form

$$y_k = \sum_{n=0}^{N} a_n x_{n-k} + \sum_{n=1}^{M} b_n y_{n-k}$$

where y_k is the output data sequence, x_k is the input data sequence, and the a_n and b_n coefficients describe the filter characteristic. Modulators are described by equations of a similar but degenerate form

$$y_k = m_k x_k$$

where m_k is the modulation sequence. Multiplexers are described by equations of the form

$$y_k = \sum_{\ell=1}^{L} n_\ell x_{k\ell}$$

where one n_ℓ at a time is unity, the others are zero and $x_{k\ell}$ is the ℓ^{th} input at time k.

Image processing requirements are frequently of the form

$$y_{mn} = \sum_{j} \sum_{k} a_{jk} x_{m-j,\, n-k}$$

as are many forms of one and two-dimensional transforms. This primal computational form has arisen so frequently in signal-processing systems that the efficient mechanization of this computation is seen as a vital step in DoD's quest for high-speed Signal processing.

Mechanization of signal-processing functions is painfully slow when faced with high-speed data for real-time processing. Digital signal-processing systems can be improved in terms of speed and lower cost if the normally employed arithmetic operations are streamlined.

For instance, a high resolution 875-line video system with a 4:3 aspect ratio and 8-bit amplitude resolution provides about 8.17 million bits per frame. At a frame rate of 60 per second, a 3-color display now requires a data rate of 1.47 Gigabits/sec! Image enhancement calculations at that speed seem impossible by today's standards.

Special signal-processing devices and organizations must be developed to meet the increasingly difficult signal-processing requirements. The development of the programmable image-processing element is an effort to evolve a programmable microprocessor type of device which is specifically a signal processor for high data rate applications.

The most frequently required image-processing function is that of a sliding window. The input is a 3x3 array of picture elements, or pixels. Each pixel is weighted by a coefficient, and the sum of products is the output, usually assigned to a location in output space which corresponds to the center of the 3x3 array.

Edge enhancement can be performed with an approximation to the bi-Laplacian operator, $\frac{\partial^4(\cdot)}{\partial x^2 \partial y^2}$ as described in reference 1:

$$\nabla = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

For directional edge information there exist several means to approximate the partial derivative in the direction of interest as shown in the following figures 1 and 2 which has been taken from reference 2.

These masks are but a sampling of the variety of window functions. Integration can also be performed by approximating a sidelobe-suppressing mean operator:

$$M_0 = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

The purpose of this section has been to illustrate the versatility of and the need for inner-product operators for signal processing.

Direction of Edge	Direction of Gradient	Prewitt Masks	Kirsch Masks	Three-level Simple Masks	Five-level Simple Masks
0	North	$\begin{bmatrix}1&1&1\\1&-2&1\\-1&-1&-1\end{bmatrix}$	$\begin{bmatrix}5&5&5\\-3&0&-3\\-3&-3&-3\end{bmatrix}$	$\begin{bmatrix}1&1&1\\0&0&0\\-1&-1&-1\end{bmatrix}$	$\begin{bmatrix}1&2&1\\0&0&0\\-1&-2&-1\end{bmatrix}$
1	Northwest	$\begin{bmatrix}1&1&1\\1&-2&-1\\1&-1&-1\end{bmatrix}$	$\begin{bmatrix}5&5&-3\\5&0&-3\\-3&-3&-3\end{bmatrix}$	$\begin{bmatrix}1&1&0\\1&0&-1\\0&-1&-1\end{bmatrix}$	$\begin{bmatrix}2&1&0\\1&0&-1\\0&-1&-2\end{bmatrix}$
2	West	$\begin{bmatrix}1&1&-1\\1&-2&-1\\1&1&-1\end{bmatrix}$	$\begin{bmatrix}5&-3&-3\\5&0&-3\\5&-3&-3\end{bmatrix}$	$\begin{bmatrix}1&0&-1\\1&0&-1\\1&0&-1\end{bmatrix}$	$\begin{bmatrix}1&0&-1\\2&0&-2\\1&0&-1\end{bmatrix}$
3	Southwest	$\begin{bmatrix}1&-1&-1\\1&-2&-1\\1&1&1\end{bmatrix}$	$\begin{bmatrix}-3&-3&-3\\5&0&-3\\5&5&-3\end{bmatrix}$	$\begin{bmatrix}0&-1&-1\\1&0&-1\\1&1&0\end{bmatrix}$	$\begin{bmatrix}0&-1&-2\\1&0&-1\\2&1&0\end{bmatrix}$
4	South	$\begin{bmatrix}-1&-1&-1\\1&-2&1\\1&1&1\end{bmatrix}$	$\begin{bmatrix}-3&-3&-3\\-3&0&-3\\5&5&5\end{bmatrix}$	$\begin{bmatrix}-1&-1&-1\\0&0&0\\1&1&1\end{bmatrix}$	$\begin{bmatrix}-1&-2&-1\\0&0&0\\1&2&1\end{bmatrix}$
5	Southeast	$\begin{bmatrix}-1&-1&1\\-1&-2&1\\1&1&1\end{bmatrix}$	$\begin{bmatrix}-3&-3&-3\\-3&0&5\\-3&5&5\end{bmatrix}$	$\begin{bmatrix}-1&-1&0\\-1&0&1\\0&1&1\end{bmatrix}$	$\begin{bmatrix}-2&-1&0\\-1&0&1\\0&1&2\end{bmatrix}$
6	East	$\begin{bmatrix}-1&1&1\\-1&-2&1\\-1&1&1\end{bmatrix}$	$\begin{bmatrix}-3&-3&5\\-3&0&5\\-3&-3&5\end{bmatrix}$	$\begin{bmatrix}-1&0&1\\-1&0&1\\-1&0&1\end{bmatrix}$	$\begin{bmatrix}-1&0&1\\-2&0&2\\-1&0&1\end{bmatrix}$
7	Northeast	$\begin{bmatrix}1&1&1\\-1&-2&1\\-1&-1&1\end{bmatrix}$	$\begin{bmatrix}-3&5&5\\-3&0&5\\-3&-3&-3\end{bmatrix}$	$\begin{bmatrix}0&1&1\\-1&0&1\\-1&-1&0\end{bmatrix}$	$\begin{bmatrix}0&1&2\\-1&0&1\\-2&-1&0\end{bmatrix}$

Figure 1. Examples of Compass Gradient Masks

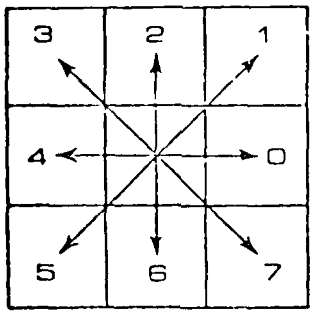


Figure 2. The Eight Principal Directions on a 3x3 Grid

G.S. Robinson, "Detection and Coding of Edges Using Directional Masks," SPIE Vol. 87, Advances in Image Transmission Techniques (1976) pp. 117-125.

3. Recommended Architecture

3.1 Principle of Operation

In the preceding chapter we demonstrated that our fundamental premise is that most signal-processing tasks can be expressed as a vector dot (or inner, or scalar) product,

e.g.: $$p = x_1 y_1 + x_2 y_2 + \dots\dots x_J y_J$$

$$= \sum_{J=1}^{J} x_j y_j$$

$$= \underset{\sim}{x}^T \underset{\sim}{y}$$

where $\underset{\sim}{x} = \text{col}\left[x_1, x_2, \dots\dots x_j\right]$

and $\underset{\sim}{y} = \text{col}\left[y_1, y_2, \dots\dots y_j\right]$

As Peled and Liu observed,[3] if we consider the x_j as being composed of numbers of K amplitude bits and a sign bit, the x_j can be expressed as fractional values:

$$x_j = \sum_{k=0}^{K} a_{jk} 2^{-k}$$

Therefore, the inner product may be written as

$$p = \sum_{j=1}^{J} \sum_{k=0}^{K} a_{jk} y_j 2^{-k}$$

$$= \sum_{k=0}^{K} \left[\sum_{j=1}^{J} a_{jk} y_j \right] 2^{-k}$$

$$= \sum_{k=0}^{K} q_k 2^{-k}$$

where $$q_k = \sum_{j=1}^{J} a_{jk} y_j$$

The above expressions imply two facts:

Each q_k may be generated by <u>one</u> table lookup operation in a 2^J-1 word memory, where the word length, $W = [\log_2 J]$ + requirement on individual y_j.

p may be generated by K shift-and-add (S&A) operations.

These equations can be mechanized by the simple configuration which is shown below,

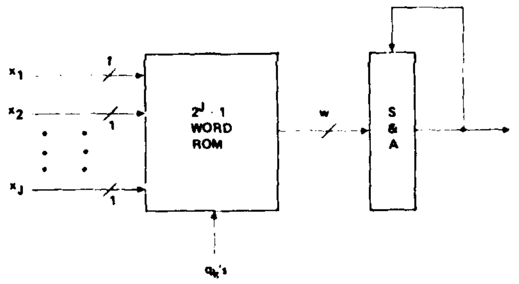


Figure 3. Basic ROM-Accumulator Structure

where the organization of the ROM is indicated below in Table 1 for the J=3 case.

TABLE 1. BASIC MEMORY STRUCTURE & ORGANIZATION

INPUT PATTERN EQUALS MEMORY ADDRESS			MEMORY CONTENT
X_1	X_2	X_3	q_k
0	0	0	0
0	0	1	y_3
0	1	0	y_2
0	1	1	y_2+y_3
1	0	0	y_1
1	0	1	y_1+y_3
1	1	0	y_1+y_2
1	1	1	$y_1+y_2+y_3$

If our problem were initially one of convolution, such as

$$P_h = \sum_{j=1}^{J} x_{h-j}\, y_j$$

then

$$P_h = \sum_i \sum_j \sum_k a_{h-j,k}\, b_{h-j,i-k} 2^{-i}$$

which is two-dimensional convolution. Three facts can be drawn out from this:

1. Looking at the problem as one of bit manipulation raises the order of convolution by only one. (The generalization can be rigorously justified.)
2. Convergence is uniform so we can operate the summations in any order without affecting the answer.
3. This impacts heavily computing time and complexity of hardware.

Let's return our attention to the basic inner-product statement:

$$p = \sum_i \sum_j \sum_k a_{jk}\, b_{j,i-k} 2^{-i}$$

We could mechanize this by summing over j, then over k

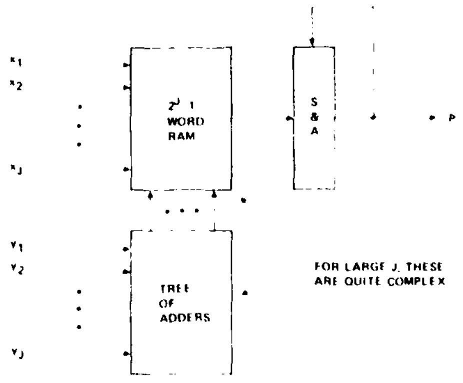


Figure 4. Block Multiplier Mechanization

or we could mechanize by summing this over k, then over j

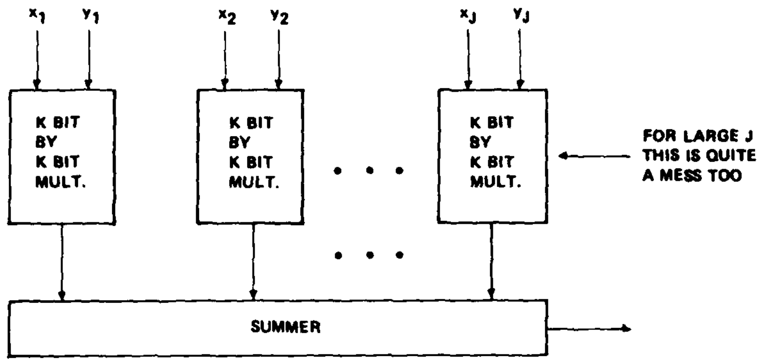


Figure 5 Conventional Mechanization

Neither solution is satisfactory. Each solution represents an extreme. If we explore summing over part of j, several times, then over k, then our remainder of j (which we can do because the order of operations does not affect the answer), we have in interesting hybrid solution.

3.2 The Search for an Optimum Answer[4]

Suppose

1. We use M table-lookup multipliers
2. Each block multiplier operates on vectors of dimension N

Since the total dimension of the vectors in the problem is J, then J=MN. A product accuracy of K + 1 bits will be maintained, so

$$p = \sum_{j=1}^{J} x_j y_j = \sum_{m=1}^{M} p_m$$

where p_m is the output of the $m^{\underline{th}}$ block multiplier

$$p_m = \sum_{n=1}^{N} x_{n+N(m-1)} \, y_{n+N(m-1)}$$

and each x can be decomposed into its constituent bits:

$$x_{n+N(m-1)} = \sum_{k=0}^{K} a(n,m,N,k)2^{-k}$$

The new system which we are examining is shown below in Figure 6.

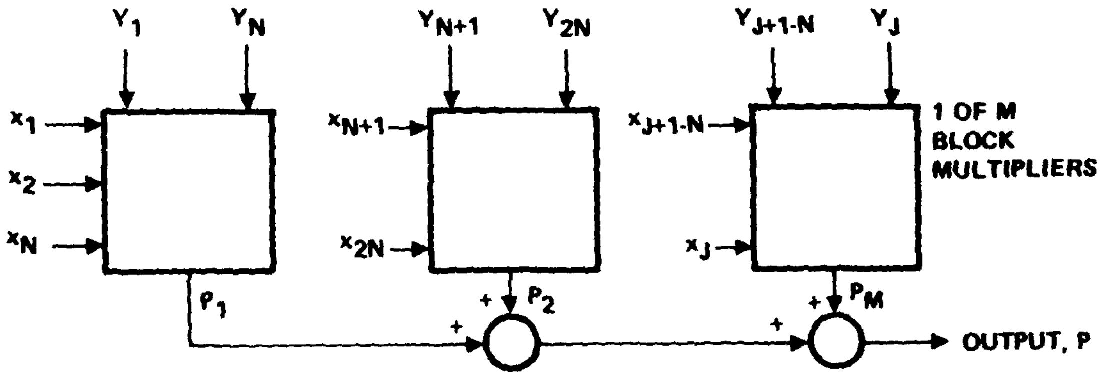


Figure 6 Candidate Hybrid Configuration for Inner-Product Generation

3. Expanding the Versatility

We have much more powerful processing means if both x and y components are free variables. Let's go back to the inner-product statement

$$p = \sum_{j=1}^{J} x_j y_j$$

where x was composed of K + 1 bits:

$$x_j = \sum_{k=0}^{K} a_{jk} 2^{-k} \quad .$$

Now y can be similarly composed of L + 1 bits:

$$y_j = \sum_{\ell=0}^{L} b_{j\ell} 2^{-\ell}$$

so

$$p = \sum_j \sum_k \sum_\ell a_{jk} \, b_{j\ell} 2^{-(k+\ell)} \quad .$$

If we define a new exponent of 2

$$i \triangleq k+\ell$$

the inner-product statement becomes

$$p = \sum_i \sum_j \sum_k a_{jk} \, b_{j,i-k} 2^{-i}$$

which is one-dimensional convolution.

Now we need a computational-complexity measure.

A minimum-complexity mechanization for a serial/parallel multiplier requires K full adders, plus some overhead logic. We will use this to construct our computational complexity measure.

Conventional mechanization of $p = \sum_{j=1}^{J} x_j y_j$ therefore requires

$K \frac{\text{adders}}{\text{product}}$ x J products + [J-1] adders to sum products = KJ+J-1 adders.

A block multiplier requires $\frac{\text{K adders}}{\text{block mult.}}$ for the shift-and-add section plus the adder tree.

The number of adders required to combine N inputs into sets of S is $\binom{N}{S}$, therefore, the total number of adders in a block multiplier structure is

$$\sum_{s=2}^{N} \binom{N}{S} = \sum_{s=0}^{N} \binom{N}{S} - (N+1) = 2^N - (N+1).$$

The total number of adders for the hybrid collection-of-block-multipliers approach is

$[K+2^N-(N+1)] \frac{\text{adders}}{\text{multiplier}}$ x M multipliers + [M-1] adders to sum products.

Table 2 explores the boundary conditions of mechanization approach vs the dimension of the vector for two sample word lengths; 8 bits and 16 bits are explored as an example.

Table 2 - BOUNDARY CONDITIONS

COMPARISON OF MULTIPLIER COMPLEXITY
(NUMBER OF ADDERS REQUIRED)

Dimension of Vectors, J	M=1; N=J Block Multiplier $K+2^J-(J+1)$		M=J; N=1 Conventional $J(K+1)-1$	
	8 Bits	16 Bits	8 Bits	16 Bits
2	8	16	15	31
3	11	19	23	47
4	18	26	31	63
5	33	41	39	79
6	64	72	47	95
7	127	135	55	111
8	254	262	63	127
9	509	517	71	143
10	1020	1028	79	159

We see that an optimum solution lies between $1<M<J$ and $1<N<J$.
The number of adders used in one hybrid solution is:

$$A = [K+2^N - (N+1)]\, M+[M-1] = [K+2^N-N]\, M-1$$

Our formal minimization procedure is the following:

Normalize with respect to $J = MN$

Define the total number of bits as $B = K+1$

For computational convenience, define the auxiliary variable

$$\hat{A} = \frac{A+1}{J}$$

$$= \frac{B-1+2^N}{N} -1$$

and minimize $\hat{A}$ with respect to N. Now, let us compare this result with that of conventional mechanization:

$$A_c = KJ+J-1 = (B-1)J+J-1 = BJ-1$$

For computational simplicity we will describe another auxiliary variable:

$$\hat{A}_c = \frac{A_c+1}{J} = B$$

Notice that we cannot compare this with straight block multiplier mechanization using the same formulation because the parameter J cannot be removed by normalizing, i.e.:

$$A_B = K+2^J - (J+1) = B+2^J - J-2$$

$$\hat{A}_B = \frac{A_B+1}{J} = \frac{B+2^J -1}{J} -1$$

A comparison of $\hat{A}$ for various values of B and N is shown in Table 3. The N=1 column is identically the same as $\hat{A}_c$. For word lengths of 4 bits the optimum combination is 2-element vectors; for word lengths of 8, 12, and 16 bits, the optimum combination is 3-element vectors; for word lengths of 20, 24, 28, 32, and 36 bits the optimum combination is 4-element vectors.

TABLE 3. $\hat{A}$ FOR VARIOUS B, N VALUES

B \ N	1(=A_c)	2	3	4	5
4	4	2 1/2	2 2/3	3 3/4	6
8	8	4 1/2	4	4 3/4	6 4/5
12	12	6 1/2	5 1/3	5 3/4	7 3/5
16	16	8 1/2	6-2/3	6-3/4	8-2/5
20	20	10 1/2	8	7-3/4	9-1/5
24	24	12 1/2	9-1/3	8-3/4	10
28	28	14 1/2	10-2/3	9-3/4	10-4/5
32	32	16 1/2	12	10-3/4	11-3/5
36	36	18 1/2	13-1/3	11-3/4	12-2/5

Shown in Table 4 is a tabulation of $\hat{A}_B$ for various values of B and J. Not suprisingly, if J is replaced by N, Table 4 contains the same numerical entries as Table 3, validating again our conclusion of what constitutes a minimum-complexity multiplier.

TABLE 4. $\hat{A}_B$ (B,J)

B \ J	2	3	4	5	6	7	8	9	10
4	2½	2-2/3	3-3/4	6	10-1/6	17-5/7	31 3/8	56-2/9	101-7/10
8	4½	4	4-3/4	6-4/5	10-5/6	18-2/7	31-7/8	56-2/3	102-1/10
12	6½	5-1/3	5-3/4	7-3/5	11-1/2	18-6/7	32-3/8	57-1/9	102-1/2
16	8½	6-2/3	6-3/4	8-2/5	12-1/6	19-3/7	32-7/8	57-5/9	102-9/10
20	10½	8	7-3/4	9-1/5	12-5/6	20	33-3/8	58	103-3/10
24	12½	9-1/3	8-3/4	10	13-1/2	20-4/7	33-7/8	58-4/9	103-7/10
28	14½	10-2/3	9-3/4	10-4/5	14-1/6	21-1/7	34-3/8	58-8/9	104-1/10
32	16½	12	10-3/4	11-3/5	14-5/6	21-5/7	34-7/8	59-1/3	104-1/2
36	18½	13-1/3	11-3/4	12-2/5	15-1/2	22-2/7	35-3/8	59-7/9	104-9/10

3.4 A Solution Based on Applications

The following set of observations appears to be valid:

For word lengths of 8, 12, and 16 bits, 3x3 table lookup multipliers are most efficient.

For word lengths greater than 16 bits, 4x4 table lookup multipliers are most efficient. If, however, we used the 3x3 table lookup multipliers for the greater word length cases, the percentages of extra adders (over the optimum 4x4) is

TABLE 5 ADDER REQUIREMENTS
FOR 3x3 ORGANIZATION

WORD LENGTH, B	% OF ADDITIONAL NORMALIZED ADDERS
20	3%
24	7%
28	9%
32	12%
36	13%

We may therefore conclude that the penalty is small for the advantages gained by using the standardized 3x3 table lookup multiplier structure.

On the other hand, if we use the 4x4 table-lookup multiplier for the shorter word lengths, the penalty paid in terms of additional adders over the 3x3 table-lookup multiplier is shown below in Table 6.

TABLE 6 ADDER REQUIREMENTS
FOR 4x4 ORGANIZATION

WORD LENGTH, B	% OF ADDITIONAL NORMALIZED ADDERS
4	41
8	19
12	8
16	1

We have to look now at applications in order to decide if we should use a 3x3 multiplier as standard or a 4x4 multiplier.

The 4x4 multiplier is a natural building block to perform a 2D transform of dimension $2^N \times 2^N$ where N is an integer.

The 4x4 multiplier is also the natural building block to perform a complex FFT butterfly computation. The decimate-in-frequency butterfly is described by

$$x = u + v$$
$$y = wz$$
$$z = u - v$$

where the complex inputs are u and v, the complex outputs are x and y, the complex weighting coefficient is w, and z is an auxiliary variable. Then

$$x_r = u_r + v_r$$
$$x_i = u_i + v_i$$
$$y_r = u_r w_r - u_i w_i - v_r w_r + v_i w_i$$
$$y_i = u_r w_i + u_i w_r - v_r w_i - v_i w_r$$

or

$$\begin{bmatrix} x_r \\ x_i \\ y_r \\ y_i \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ w_r & -w_i & -w_r & w_i \\ w_i & w_r & -w_i & -w_r \end{bmatrix} \begin{bmatrix} u_r \\ u_i \\ v_r \\ v_i \end{bmatrix}$$

which can be performed handily by our 4x4 structure. The decimate-in-time butterfly is described by

$$x = u + p$$
$$y = u - p$$
$$p = vw$$

where u, v, w, x, and y are as described above and p is an auxiliary variable.

$$x_r = u_r + v_r w_r - v_i w_i$$
$$x_i = u_i + v_r w_i + v_i w_r$$
$$y_r = u_r - v_r w_r + v_i w_i$$
$$y_i = u_i - v_r w_i - v_i w_r$$

or

$$\begin{bmatrix} x_r \\ x_i \\ y_r \\ y_i \end{bmatrix} \begin{bmatrix} 1 & 0 & w_r & -w_i \\ 0 & 1 & w_i & w_r \\ 1 & 0 & -w_r & w_i \\ 0 & 1 & -w_i & w_r \end{bmatrix} \begin{bmatrix} u_r \\ u_i \\ v_r \\ v_i \end{bmatrix}$$

which again can be performed handily by the 4x4 structure.

A set of 4 such vector multipliers can also be used directly as a general 4x4 coordinate transformer or as a matrix multiplier, or to mechanize a filter as described below in Table 7.

TABLE 7 FUZZ-PHRASE FILTER LIST

(Pick a line from ach category, A thru G, to describe function)

A. 1. Fixed Coefficient 2. Programmed time-variable parameter 3. Adaptive	E. 1. One-Dimensional 2. Two-Dimensional
B. 1. Linear 2. Programmed Nonlinear	F. 1. Low-pass 2. High-pass 3. Bandpass 4. Band reject 5. All pass 6. Arbitrary spectrum shaping
C. 1. Recursive 2. Transversal	
D. 1. Fixed Structure 2. Variable Order (up to 16 taps)	G. 1. Filter 2. Equalizer

The same identical architecture can be programmed to also function as a

1. Modulator/Demodulator (set of 4)
2. Coordinate Transformer (up to 4 rotations)
3. Polynomial Function Generator
4. Element of Pattern Classifier
5. Multiplexer (4-4:1 or 2-8:1 or 1-8:1 and 2-4:1 or 1-12:1 and 1-4:1 or 1-16:1)
6. Matched Filter
7. Edge-Extraction Mask
8. Sobel Operator
9. Cosine Transformer (4x4, or 8x2, or 16x1)
10. Hadamard Transformer
11. Unsharp Masking
12. Despike Element
13. Etc.

For Operations in: Mapmatching
Midcourse Updating
Doppler Radar and processing
Target detection, identification, tracking, cueing
Aimpoint selection
Correlation
Data windowing
Filtering of signals
Sonar spectrum analysis
Inertial platform stabilization
Instrument caging
Flight control stability augmentation
Adaptive noise cancelling
Speech enhancement
Adaptive line enhancement/cancellation
Channel equalization for data modems
Data compaction/AJ protection
FLIR Display Systems
Dynamic range compression functions
Autothresholding for video noise limiting
Chrome separation for digital video
Pattern recognition systems
Automatic fingerprint classification
Optical character reader
Nonlinear noise filter
Thin-fill 2D data reduction
.
.
.

3.5 A Candidate Element

The skeleton of a basic signal processing section based on the foregoing is shown in Figure 7. This circuit mechanizes in each of 4 sections the following function

$$y_k = \sum_{j=0}^{3} a_{jk} x_{jk}$$

where the x_{jk} may be independent (general inner product), or successive samples of an input (transversal filter) or some may be successive samples of the output (recursive filter). The coefficients a_{jk} may be wholly replaced each sample time, or incrementally updated (adaptive filter).

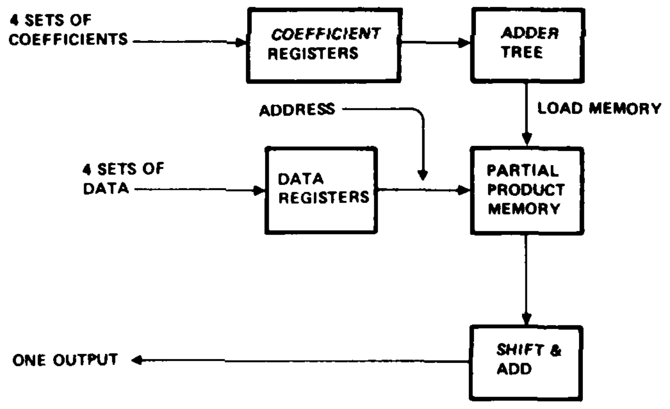


Figure 7 Skeleton of Signal Processing Section
(1 of 4 sections on a chip)

The 4 data words address the partial-product memory as in section 3-1. The contents of the partial-product memory were obtained by combining the coefficients through the adder tree.

Büttner and Schüssler[5] have shown that since x_{jk} can be expressed in terms of its N bits, b_{jkm}, then for 2's-complement format,

$$x_{jk} = -b_{jk0} + \sum_{n-1}^{N-1} b_{jkn}\, 2^{-n}$$

and since

$$x_{jk} = \frac{1}{2}\, x_{jk} - \frac{1}{2}\,(-x_{jk}) = \frac{1}{2}(b_{jk0} - \bar{b}_{jk0}) + \frac{1}{2} \sum_{n=1}^{N-1} (b_{jkn} - \bar{b}_{jkn}) 2^{-n} - 2^{-N}$$

then

$$y_k = \frac{1}{2}\, [-q(0)\bar{2}^{(N-1)} + \sum_{n=1}^{N-1} q(\{b_{jkn}\}) - (b_{jk0} - \bar{b}_{jk0})\; q(\{b_{jko}\})]$$

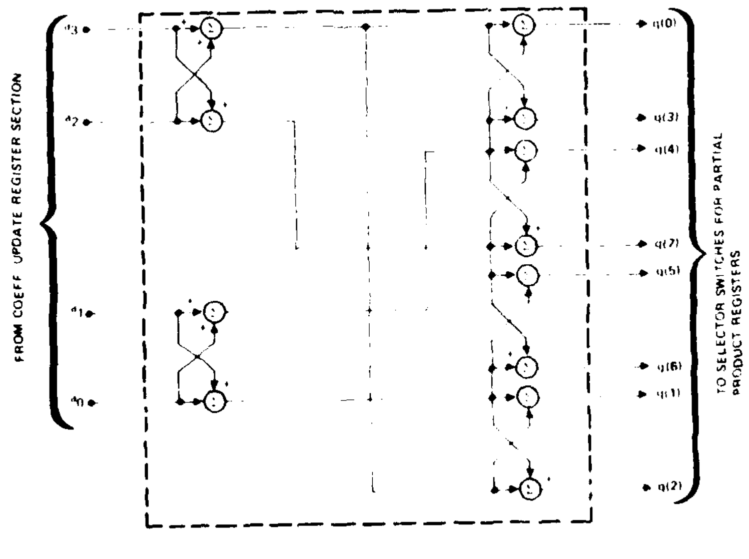
where

$$q(n) = q(\{b_{jkn}\}) = \sum_{j=0}^{3} a_{jk}(b_{jkn} - \bar{b}_{jkn}) \qquad \text{for } n \in \left[0,\ 7\right]$$

and

$$q(n) = -q(15-n)\ . \qquad \text{for } n \in \left[8,\ 15\right]$$

An extremely efficient streamlined mechanization is shown below. The a's are combined to generate all possible q(n) as shown in the adder tree of Figure 8.



FROM COEFF UPDATE REGISTER SECTION
a3
a2
a1
a0
q(0)
q(3)
q(4)
q(7)
q(5)
q(6)
q(1)
q(2)
TO SELECTOR SWITCHES FOR PARTIAL PRODUCT REGISTERS

Figure 8. Adder-tree Section

The a's are 16-bit numbers, but the q(n) are 18-bit numbers since q(n) can be as large as the sum of 4 a's. We want 3 sets of "q" registers; one as the momentary working set, and one or two being loaded as shown in Figure 9.

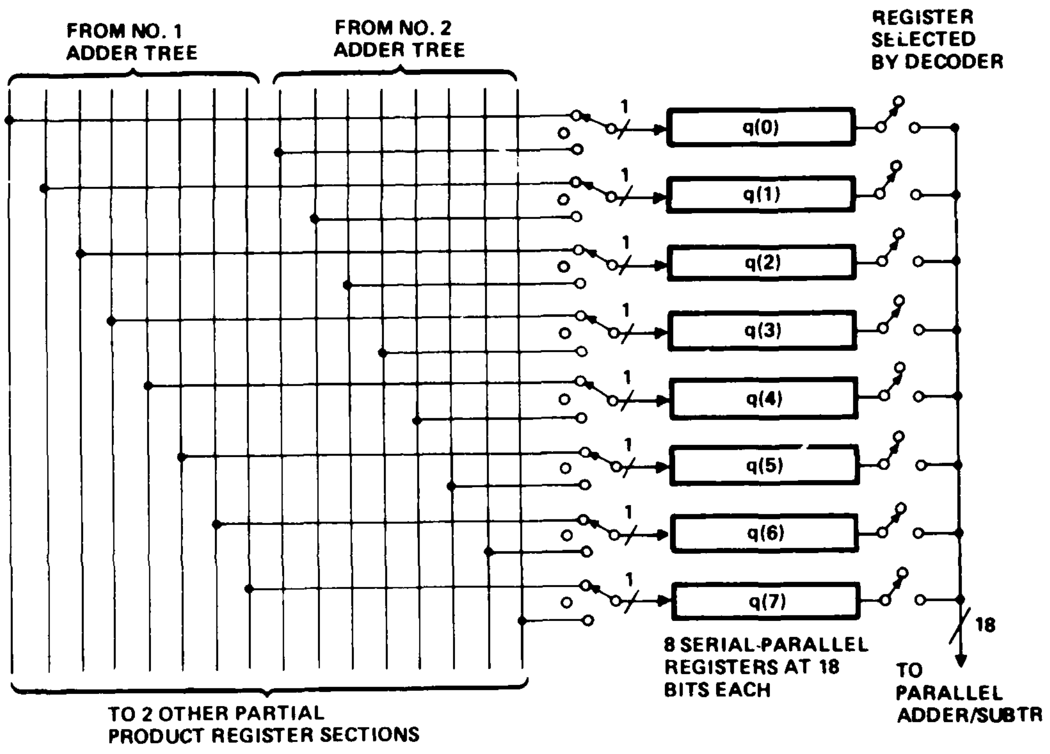

FROM NO. 1 ADDER TREE
FROM NO. 2 ADDER TREE
REGISTER SELECTED BY DECODER
1
q(0)
q(1)
q(2)
q(3)
q(4)
q(5)
q(6)
q(7)
18
8 SERIAL-PARALLEL REGISTERS AT 18 BITS EACH
TO PARALLEL ADDER/SUBTR
TO 2 OTHER PARTIAL PRODUCT REGISTER SECTIONS

Figure 9. Partial-Product Register Section

The outputs which feed the adder/subtractor are then summed (or differenced) according to $y_k = \Sigma q(n)2^{-n}$.

The selection of a particular q(n) is according to the bits in x_{jk}, i.e., the b_{jkn}. The decoding network (partial-product register selector) is shown in Figure 10.

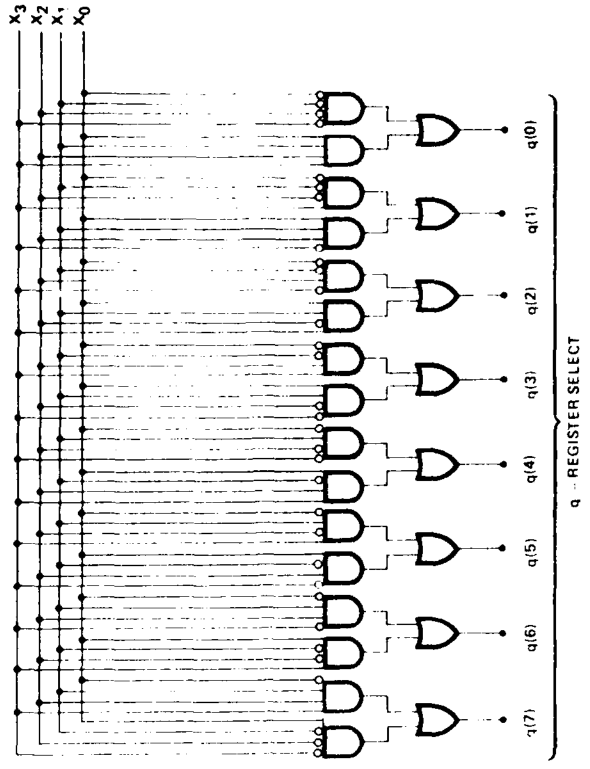


Figure 10. Partial-Product Register Selector

The circuit of Figure 10 is driven by the data-input-register section which is shown in Figure 11. The 4 shift registers may each be driven from a serial input path, chained with the preceding register or loaded from the output. These 3 options for each of 4 registers give 12 control states which are addressed by 4 function-select lines.

The x outputs from the shift registers may also be used to provide the "internal" signals to the coefficient update register section which is shown in Figure 12. This is to facilitate the adaptive filter update computation

$$a_{jk} = a_{j(k-1)} + \mu x_{jk} \mathrm{sgn}(\varepsilon_k).$$ [6]

The μ is mechanized by the shift. Since the update must be formed before the error signal, ε_k, is available, one update-register section assumes $\varepsilon_k > 0$, the other assumes $\varepsilon_k < 0$. The correct q-register set is chosen after ε_k has been completed. This is the reason for two sets of registers being loaded simultaneously.

A basic signal-processing section based upon the foregoing is shown in Figure 13.

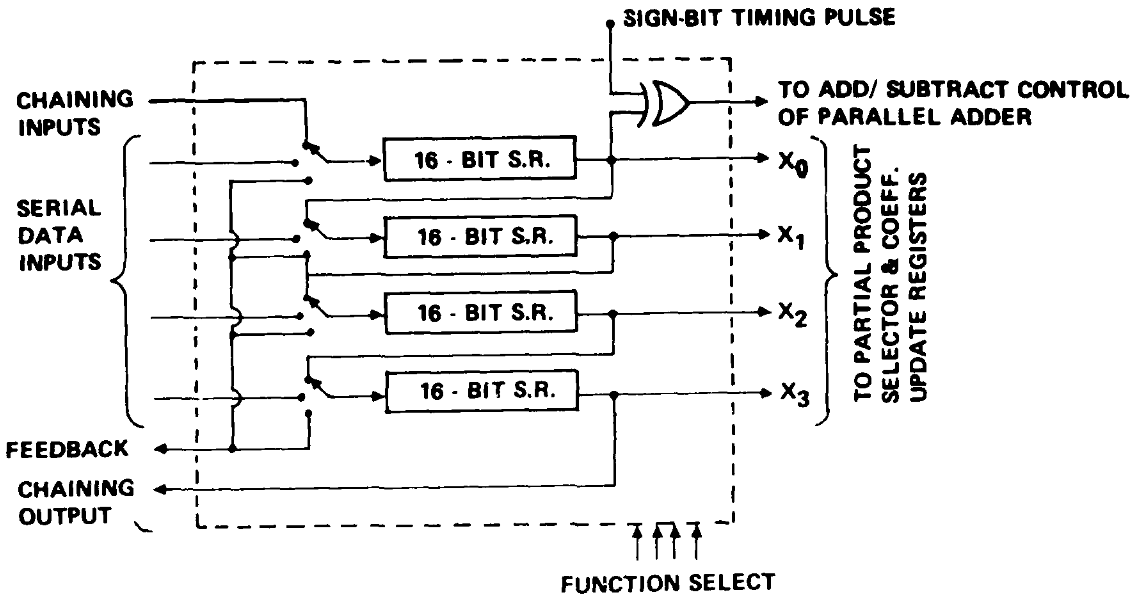


Figure 11. Data-Input Register Section

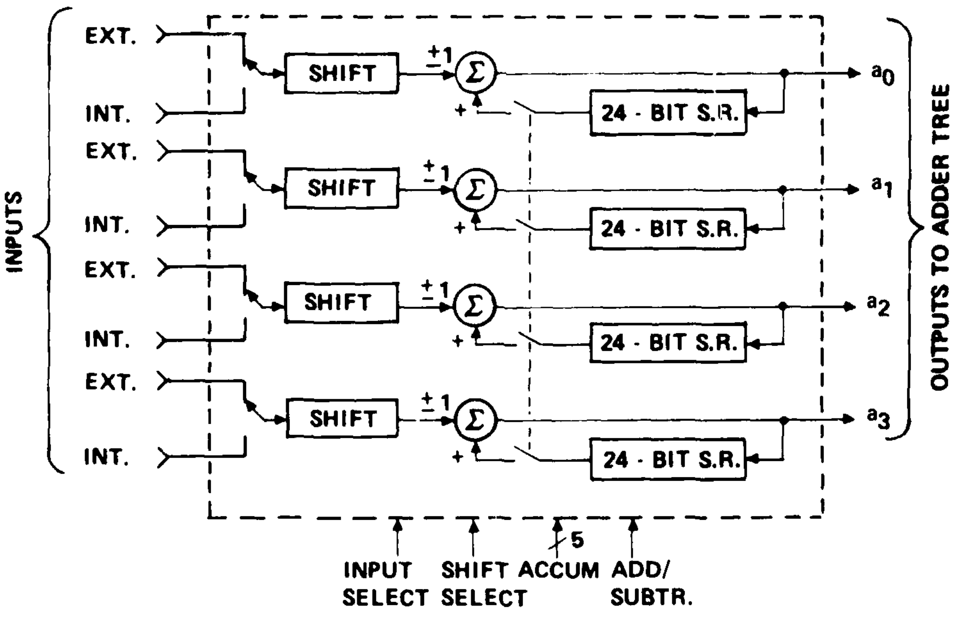
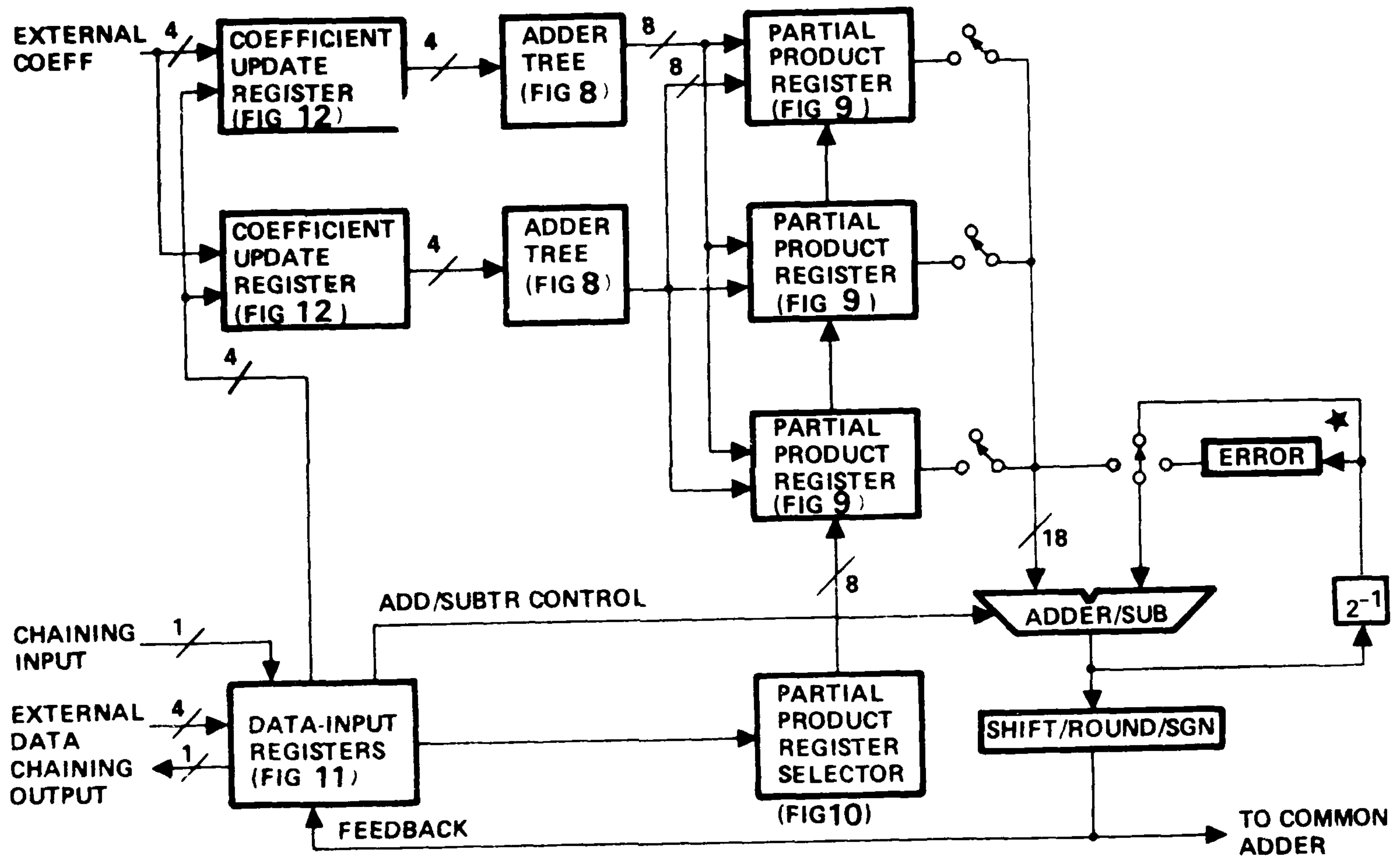


Figure 12. Coefficient-Update-Register Section


EXTERNAL COEFF
4
COEFFICIENT UPDATE REGISTER (FIG 12)
4
ADDER TREE (FIG 8)
8
8
PARTIAL PRODUCT REGISTER (FIG 9)
COEFFICIENT UPDATE REGISTER (FIG 12)
4
ADDER TREE (FIG 8)
PARTIAL PRODUCT REGISTER (FIG 9)
4
PARTIAL PRODUCT REGISTER (FIG 9)
ERROR
18
8
ADD/SUBTR CONTROL
ADDER/SUB
2^{-1}
CHAINING INPUT
1
EXTERNAL DATA
4
CHAINING OUTPUT
1
DATA-INPUT REGISTERS (FIG 11)
PARTIAL PRODUCT REGISTER SELECTOR (FIG10)
SHIFT/ROUND/SGN
FEEDBACK
TO COMMON ADDER

*Control roundoff error buildup. See (7), T. Chang, "A Low Roundoff-Noise Digital Filter Structure," Proc. IEEE ISCAS, May 1978.

Figure 13. Signal Processing Section (1 of 4 Sections on a Chip)

Figure 14 shows how the signal-processing sections of Figure 13 are interconnected to extend the upper limit of the y_k sum from 3 to a number as great as 15 on a single chip.

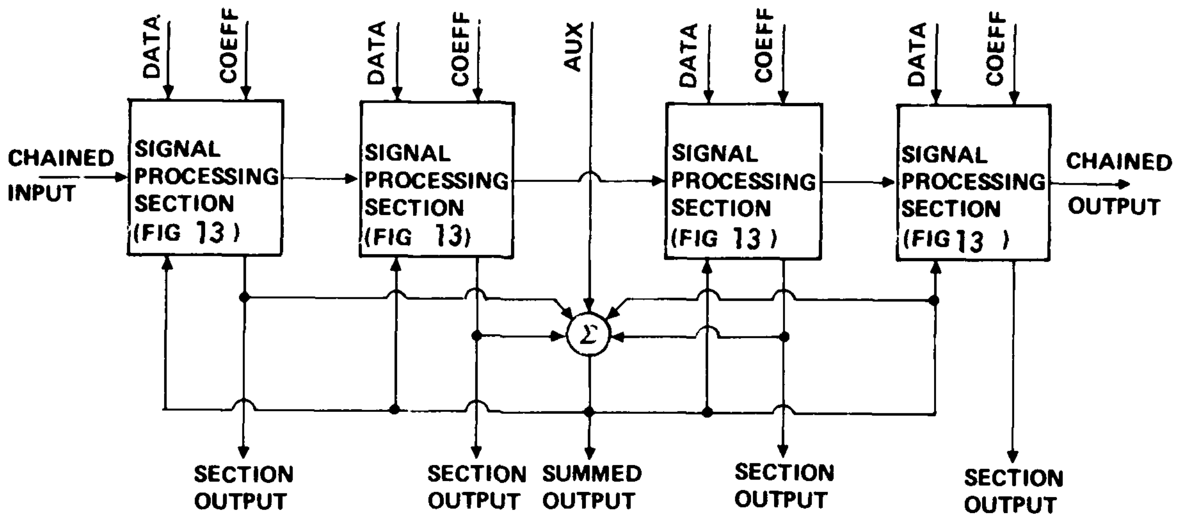

DATA
COEFF
DATA
COEFF
AUX
DATA
COEFF
DATA
COEFF
CHAINED INPUT
SIGNAL PROCESSING SECTION (FIG 13)
SIGNAL PROCESSING SECTION (FIG 13)
SIGNAL PROCESSING SECTION (FIG 13)
SIGNAL PROCESSING SECTION (FIG 13)
CHAINED OUTPUT
Σ
SECTION OUTPUT
SECTION OUTPUT
SUMMED OUTPUT
SECTION OUTPUT
SECTION OUTPUT

Figure 14. Total Signal-Processing Device

The functions which we have discussed above are capable of mechanizing signal-processing requirements such as:

- Vector-matrix operator with fixed or variable coefficients 4 x 4, or 2 x 8, or 1 x 8 and 2 x 4, or 1 x 12 and 1 x 4 or 1 x 16 (dimensions may be raised by chaining with other devices.)

- Generalized fast-generalized-transform operator (decimate in time or decimate in sequency).

- Digital filter (up to 4)
  fixed parameter, variable parameter or adaptive
  denominator order, $D \in \{0,15\}$
  numerator order, $N \in \{0,15-D\}$

Set of 4 modulators/demodulators

Multiplexers

4-4:1 or 2-8:1 or 1-8:1 and 2-4:1 or 1-12:1 and 1-4:1 or 1-16:1

Image-processing functions such as sliding windows

3.6 Problems and Retrenchment

A preliminary transistor count revealed that this very desirable structure would be an extremely ambitious circuit with 88,000 transistors. The device design could not be completed under the contract.

A second, less ambitious, structure which was a nine-element vector multiplier was pursued into the design and layout stages. That design, which became identified as the PIPE device, is described in the following chapter.

4. The Nine-Element Vector Multiplier

4.1 Overall Description of PIPE Device

The PIPE provides every nine 10 ns-clock periods the sum of products

$$y = \sum_{n=1}^{9} a_n x_n,$$

where each a_n and x_n is an 8-bit 2's-complement number, thereby performing a true multiply-and-accumulate function 10^8 times per second. The full 19-bit product is available as an output which permits the devices to be combined to perform higher accuracy computations. The coefficient a's are parallel loaded and stored on-chip while the data x's may be loaded serially or in parallel in a fashion which makes the chip directly usable as a FIR (finite-impulse response) filter described by the transfer function

$$G(z) = \sum_{n=0}^{8} a_n z^{-n}$$

Any number of such chips can be chained together to form a longer filter. Only an external summing means is required to accumulate the final result. The device description is given below:

Supply Voltage: 5 volts

CMOS/SOS 2 μm technology using static logic

Clock Frequency: 100 MHz

Operating temperature range: -55°C to +125°C

Packaging: leadless hermetic chip carrier corresponding to JDEC specification

Input specification: there are 7 input formats; all input patterns must occur within 9 clock periods. These formats are:

1. single parallel input applied 9 (or fewer) consecutive times.
2. single parallel input applied once
3. 3 parallel inputs applied 3 times
4. 3 parallel inputs applied once

5. single serial input applied once
6. three serial inputs applied once
7. nine serial inputs applied once

The input data word is in 2's-complement, 8-bit format. The format is controlled by a 3-bit format-control line. The input section has 28 pins (3x8, 1 out, 3 control).

- Output specification:
  Single 10-bit output tristate bus for full accuracy.
  Least significant 10 bits available immediately.
  Flag indicates when the 10 most significant bits are ready.
  Most significant bits on output bus in response to external strobe.
  The output section has 14 pins (10 out, 1 flag for LSBs ready, 1 flag for MSBs ready, 1 MSB strobe, 1 reset).
- Coefficient specification:
  Single 8-bit input bus for one-at-a-time parallel loading of 8-bit 2's-complement coefficient.
  Separate 4-bit input-address identification.
  One load control, one memory-write control.
  The coefficient section has 14 pins.
- The pins required by the PIPE device are given below:

24	for 3 parallel 8-bit input data words
1	for serial data-line out
3	for input-data format
10	for parallel output data
2	for output flags
2	for output control
8	for parallel coefficient word in
4	for coefficient address
2	for coefficient control
1	for word timing
1	for clock
1	for power
1	for ground
60	pins commited

4.2 The PIPE Algorithm

Recall that the PIPE forms the product

$$y = \sum_{n=1}^{9} a_n x_n$$

where the x_n are input data words, the a_n stored coefficients, and y is the sum of the products. Both the a_n and the x_n are 8-bit, 2's-complement numbers; x_n is composed of the bits $\{b_{nm}\}$ where m = 0, 1, 2, ... 7. In order to easily describe the computational approaches which were considered, let us examine the array of bits which form the $\{x_n\}$:

x_1:	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}	b_{16}	b_{17}
x_2:	b_{20}	b_{21}	b_{22}	b_{23}	b_{24}	b_{25}	b_{26}	b_{27}
x_3:	b_{30}	b_{31}	b_{32}	b_{33}	b_{34}	b_{35}	b_{36}	b_{37}
x_4:	b_{40}	b_{41}	b_{42}	b_{43}	b_{44}	b_{45}	b_{46}	b_{47}
x_5:	b_{50}	b_{51}	b_{52}	b_{53}	b_{54}	b_{55}	b_{56}	b_{57}
x_6:	b_{60}	b_{61}	b_{62}	b_{63}	b_{64}	b_{65}	b_{66}	b_{67}
x_7:	b_{70}	b_{71}	b_{72}	b_{73}	b_{74}	b_{75}	b_{76}	b_{77}
x_8:	b_{80}	b_{81}	b_{82}	b_{83}	b_{84}	b_{85}	b_{86}	b_{87}
x_9:	b_{90}	b_{91}	b_{92}	b_{93}	b_{94}	b_{95}	b_{96}	b_{97}

Figure 15. Data-Bit Array

The combining of this array of bits with the "a" coefficients is the process by which the desired result, y, is obtained. There exist, however, several diverse means by which this combining may be accomplished. One means uses the array of bits in a row-by-row fashion. Each individual row is multiplied by the corresponding "a" coefficient and the results summed in an accumulator with the results of the previously executed products. This standard lumped-arithmetic approach requires 9 full multiplications and 8 adds into the accumulator.

The direct computation using this approach is

$$y = \sum_{n=1}^{9} y_n$$

where

$$y_n = a_n x_n$$

and

$$x_n = -b_{n0} + \sum_{m=1}^{7} b_{mn} 2^{-m}$$

so a direct computation would require that we compute 9 times

$$y_n = a_n \left[-b_{no} + \sum_{m=1}^{7} b_{mn} 2^{-m}\right]$$

or if we use the Booth algorithm[8],

$$y_n = \sum_{m=0}^{7} a_n c_{mn} 2^{-m}$$

where

$$c_{mn} = -b_{mn} + b_{mn+1,n};\; b_{8n} = 0$$

or if we use the more efficient modified-Booth (or 3 BAAT) algorithm[9],

$$y_n = \sum_{m=0}^{3} a_n d_{mn} 2^{-2m}$$

where

$$d_{mn} = [-b_{mn} + \tfrac{1}{2} b_{mn+1} + \tfrac{1}{4} b_{mn+2} + \tfrac{1}{4} b_{mn+3}]$$

One can show that 4BAAT, 5BAAT, etc., algorithms reduce to the 3BAAT cases for binary multiplication. Some improvement, but not enough.

In our quest for greater computational efficiency, we shall approach a second means which uses the array of bits of Figure 15, not in a row-by-row fashion, but in a column-by-column fashion. Interestingly, the set of bits in a column is used as a memory address. This is an adaption of the candidate element architecture which was discussed earlier.

The contents at that memory address is summed in an accumulator with one-half the previous results. This procedure requires 8 table-lookup operations and 8 adds into the accumulator, not to form each y_n, but to form the total result, y. The advantages and efficiency of this latter distributed-arithmetic method are obviously great. The following paragraphs describe the means of computation in detail.

4.3 *The Mechanics of the Algorithm*

Here's how it works. Each x_n is composed of the 8 bits, b_{nm} which combine as follows to establish the value of x_n:

$$x_n = -b_{no} + \sum_{m=1}^{7} b_{nm} 2^{-m} \tag{2}$$

The sign bit, b_{no}, is unity if x_n is a negative number, and is zero otherwise. Now, since [5]

$$x_n = 1/2[x_n -(-x_n)] \tag{3}$$

we may then express (2) as:

$$x_n - 1/2[-(b_{no} -b_{no}) + \sum_{m=1}^{7} (b_{nm} - b_{nm}) 2^{-m} -2^{-7}] \tag{4}$$

Substituting (4) into (1) yields the following expression:

$$y = 1/2 \left\{ \underbrace{-2^{-7}\left[\sum_{n=1}^{9} a_n\right]}_{\text{Initial Condition}} + \sum_{m=1}^{9} \underbrace{\left[\sum_{n=1}^{9} a_n(b_{nm} - \bar{b}_{nm})\right]}_{\text{Partial Product}} 2^{-m} \underbrace{- \sum_{n=1}^{9} a_n(b_{no} - \bar{b}_{no})}_{\substack{\text{Sign Correction} \\ \text{(m=0 term)}}} \right\} \quad (5)$$

The possible value of each b_{nm} is either 0 or 1, hence the possible value of each term $(b_{nm} - \bar{b}_{nm})$ is ± 1. The bracketed term within the "partial product" braces of (5) can take on a total of 2^9 possible values, but all entries appear twice; hence, there are only $\frac{1}{2}\ 2^9 = 256$ distinct values. If the "a" coefficients are each 8 bits in length, then each of the 256 values will be stored with an accuracy of $8+[\log_2 9]_{RU} = 12$ bits. The "sign correction" values as well as the bracketed part of the "initial condition" value happen to be among the 256 distinct values. Certainly, one valid approach to computing (5) is to use a table-lookup operation in which during the first clock period we form the first partial result:

$$r_1 = \left[-\sum_{n=1}^{9} a_n\right] + \left[\sum_{n=1}^{9} a_n(b_{n7} - \bar{b}_{n7})\right] \quad (6)$$

Both bracketed quantities were obtained from the coefficient memory. During the 2nd through 7th periods we form the 2nd through 7th partial results

$$r_p = 1/2\ r_{p-1} + \left[\sum_{n=1}^{9} a_n(b_{n,8-p} - \bar{b}_{n,8-p})\right] \quad (7)$$

where p=2 through 7. During the 8th clock period we form the final result

$$2y = r_8 = 1/2\ r_7 - \sum_{n=1}^{9} a_n(b_{no} - \bar{b}_{no}) \quad (8)$$

During the 9th clock period the result is transferred to the output register, the circuits are reinitialized, and we are ready to begin another cycle of computing *y*. A block diagram of this structure is shown in Figure 16.

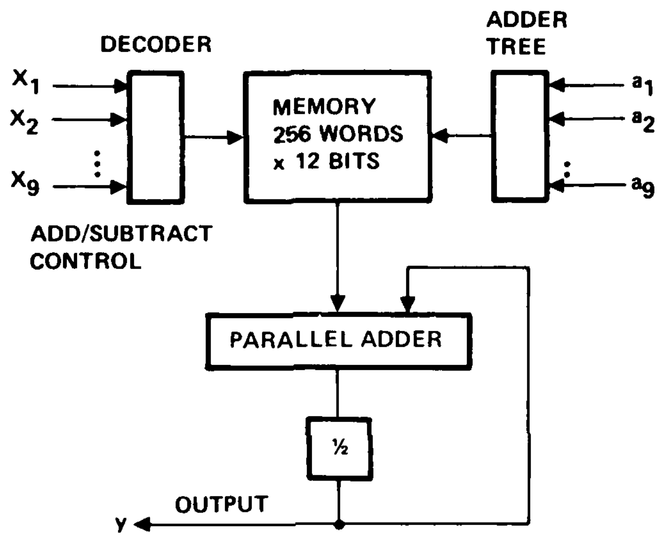


Figure 16. Basic PIPE Architecture

4-4. Modifying the Basic Algorithm

A decision was made to make the chip so that the coefficients could be changed. In order to be able to change coefficients during computation, two memory sets would be required; one being the present working memory containing functions of the "old" coefficients, the other being the memory into which we load functions of the new coefficients. Unfortunately, the resulting 2 x 256 word x 12 bit/word = 6144 bit high-speed memory was not practical to implement.

Two simplifying steps were taken. First, a restriction was established such that the coefficients could not be changed during computation. This halved the memory requirement. Secondly, we partitioned $y = y_1 + y_2$ so that

$$y_1 = \sum_{n=1}^{4} a_n x_n$$

and

$$y_2 = \sum_{n=5}^{9} a_n x_n$$

consequently,

$$2y_1 = -2^{-7} \sum_{n=1}^{4} a_n + \sum_{m=1}^{7} [\sum_{n=1}^{4} a_n (b_{nm} - \bar{b}_{nm})] 2^{-m} - \sum_{n=1}^{4} a_n (b_{no} - \bar{b}_{no})$$

and

$$2y_2 = -2^{-7} \sum_{n=5}^{9} a_n + \sum_{n=1}^{7} [\sum_{n=5}^{9} a_n (b_{nm} - \bar{b}_{nm})] 2^{-m} - \sum_{n=5}^{9} a_n (b_{no} - \bar{b}_{no})$$

Now for y_1 we need only a $1/2 \cdot 2^4$ = 8-word partial-product memory, and for y_2 we need only $1/2 \cdot 2^5$ = 16-word partial-product memory, a dramatic reduction in the number of stored words.

The 16 possible values of $\sum_{n=1}^{4} a_n (b_{nm} - \bar{b}_{nm})$ are $\pm A_0$ through $\pm A_7$ as shown in Table 8; the 32 possible values of $\sum_{n=5}^{9} a_n (b_{nm} - \bar{b}_{nm})$ are $\pm B_0$ through $\pm B_{15}$ as shown in Table 9.

TABLE 8. "A" PARTIAL PRODUCT MEMORY CONTENTS AND ADDRESSING

ADDRESS				"A" PARTIAL PRODUCT MEMORY	
b_{4m}	b_{3m}	b_{2m}	b_{1m}		
0	0	0	0	$-(a_4 + a_3) - (a_2 + a_1) = -A_0$	Only 8 Values Required
0	0	0	1	$-(a_4 + a_3) - (a_2 - a_1) = -A_1$	
0	0	1	0	$-(a_4 + a_3) + (a_2 - a_1) = -A_2$	
0	0	1	1	$-(a_4 + a_3) + (a_2 + a_1) = -A_3$	
0	1	0	0	$-(a_4 - a_3) - (a_2 + a_1) = -A_4$	
0	1	0	1	$-(a_4 - a_3) - (a_2 - a_1) = -A_5$	
0	1	1	0	$-(a_4 - a_3) + (a_2 - a_1 = -A_6$	
0	1	1	1	$-(a_4 - a_3) + (a_2 + a_1) = -A_7$	
1	0	0	0	$+(a_4 - a_5) - (a_2 + a_1) = +A_7$	Sign Change Only
1	0	0	1	$+(a_4 - a_3) - (a_2 - a_1) = +A_6$	
1	0	1	0	$+(a_4 - a_3) + (a_2 - a_1) = +A_5$	
1	0	1	1	$+(a_4 - a_3) + (a_2 + a_1) = +A_4$	
1	1	0	0	$+(a_4 + a_3) - (a_2 + a_1) = +A_3$	
1	1	0	1	$+(a_4 + a_3) - (a_2 - a_1) = +A_2$	
1	1	1	0	$+(a_4 + a_3) + (a_2 - a_1) = +A_1$	
1	1	1	1	$+(a_4 + a_3) + (a_2 + a_1) = +A_0$	

TABLE 9. "B" PARTIAL PRODUCT MEMORY CONTENTS AND ADDRESSING

ADDRESS					
b_{9m}	b_{8m}	b_{7m}	b_{6m}	b_{5m}	"B" PARTIAL PRODUCT MEMORY
0	0	0	0	0	$-a_9-(a_8 + a_7) - (a_6 + a_5) = -B_0$
0	0	0	0	1	$-a_9-(a_8 + a_7) - (a_6 - a_5) = -B_1$
0	0	0	1	0	$-a_9-(a_8 + a_7) + (a_6 - a_5) = -B_2$
0	0	0	1	1	$-a_9-(a_8 + a_7) + (a_6 + a_5) = -B_3$
0	0	1	0	0	$-a_9-(a_8 - a_7) - (a_6 + a_5) = -B_4$
0	0	1	0	1	$-a_9-(a_8 - a_7) - (a_6 - a_5) = -B_5$
0	0	1	1	0	$-a_9-(a_8 - a_7) + (a_6 - a_5) = -B_6$
0	0	1	1	1	$-a_9-(a_8 - a_7) + (a_6 + a_5) = -B_7$
0	1	0	0	0	$-a_9+(a_8 - a_7) - (a_6 + a_5) = -B_8$
0	1	0	0	1	$-a_9+(a_8 - a_7) - (a_6 - a_5) = -B_9$
0	1	0	1	0	$-a_9+(a_8 - a_7) + (a_6 - a_5) = -B_{10}$
0	1	0	1	1	$-a_9+(a_8 - a_7) + (a_6 - a_5) = -B_{11}$
0	1	1	0	0	$-a_9+(a_8 + a_7) - (a_6 + a_5) = -B_{12}$
0	1	1	0	1	$-a_9+(a_8 + a_7) - (a_6 - a_5) = -B_{13}$
0	1	1	1	0	$-a_9+(a_8 + a_7) + (a_6 - a_5) = -B_{14}$
0	1	1	1	1	$-a_9+(a_8 + a_7) + (a_6 + a_5) = -B_{15}$
1	0	0	0	0	$a_9-(a_8 + a_7) - (a_6 + a_5) = +B_{15}$
1	0	0	0	1	$a_9-(a_8 + a_7) - (a_6 - a_5) = +B_{14}$
1	0	0	1	0	$a_9-(a_8 + a_7) + (a_6 - a_5) = +B_{13}$
1	0	0	1	1	$a_9-(a_8 + a_7) + (a_6 + a_5) = +B_{12}$
1	0	1	0	0	$a_9-(a_8 - a_7) - (a_6 + a_5) = +B_{11}$
1	0	1	0	1	$a_9-(a_8 - a_7) - (a_6 - a_5) = +B_{10}$
1	0	1	1	0	$a_9-(a_8 - a_7) + (a_6 - a_5) = +B_9$
1	0	1	1	1	$a_9-(a_8 - a_7) + (a_6 + a_5) = +B_8$
1	1	0	0	0	$a_9+(a_8 - a_7) - (a_6 + a_5) = +B_7$

only 16 values required

Sign change only

TABLE 9. "B" PARTIAL PRODUCT MEMORY CONTENTS AND ADDRESSING (CONTD.)

ADDRESS					"B" PARTIAL PRODUCT MEMORY	
b_{9m}	b_{8m}	b_{7m}	b_{6m}	b_{5m}		
1	1	0	0	1	$a_9+(a_8 - a_7) - (a_6 - a_5) = +B_6$	Sign change only $\downarrow$
1	1	0	1	0	$a_9+(a_8 - a_7) + (a_6 - a_5) = +B_5$	
1	1	0	1	1	$a_9+(a_8 - a_7) + (a_6 + a_5) = +B_4$	
1	1	1	0	0	$a_9+(a_8 + a_9) - (a_6 + a_5) = +B_3$	
1	1	1	0	1	$a_9+(a_8 + a_7) - (a_6 - a_5) = +B_2$	
1	1	1	1	0	$a_9+(a_8 + a_7) + (a_6 - a_5) = +B_1$	
1	1	1	1	1	$a_9+(a_8 + a_7) + (a_6 + a_5) = +B_0$	

4-5 Mechanization of the Modified Algorithm

The PIPE consists of six basic sections: the coefficient input section, the partial-product memory section, the data input section, the arithmetic section, the output section, and the timing and control section.

4.5.1 Coefficient Input Section

The coefficients (the a's) of the defining equations of the PIPE are each 8-bit signed fractional numbers such that $-1<a<1$, the maximum value of which is $1-2^{-7}$. The partial-products (partial-product-memory content) for y_1 is from the set of numbers A_0 through A_7, the largest of whose values can be as great as $4(1-2^{-7})$. Similarly, the partial-products for y_2, from the set of numbers B_0 through B_{15}, can have values as great as

$$5(1-2^{-7}) = 2^3[1-2^{-1} -2^{-3} -2^{-7} +2^{-8} +2^{-10}]$$

an 11-bit number. We shall now examine the procedures by which these numbers are generated.

The coefficient-input section is functionally diagrammed in Figure 17. There are 9 8-bit wide, parallel registers, designated a_1 through a_9 into which the coefficients are loaded. Their inputs are wired in parallel to a common 8-bit coefficient-input bus. When the "read input" line is true, the number which is present on the 8-bit coefficient bus is loaded into the input register which was identified by the pattern on the "input-address" lines. The loading of these coefficients is completely independent of the functioning of the arithmetic section of the PIPE. However, after all the coefficients which are to be loaded into the PIPE have been loaded (by the procedure which was described above) then the PIPE is asked to "digest" these values. When the "load memory" input is true, a sequence of events is initiated which is indicated in the timing diagram of Figure 18. First, coefficient registers a_1 through a_4 parallel-load the shift registers 1 through 4. The outputs of the shift registers are clocked through complementers to generate the negative of the numbers. Each complementer

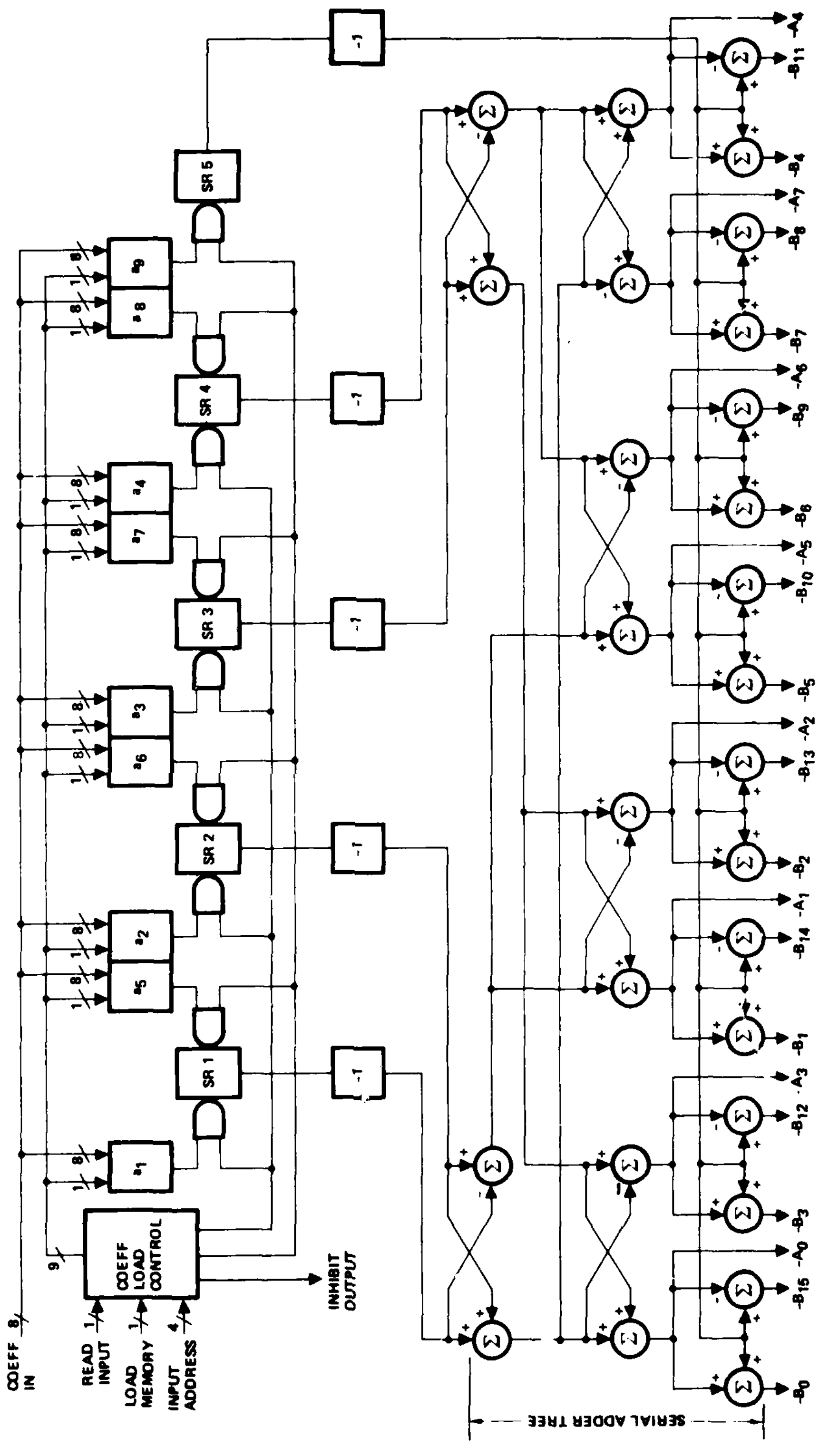


Figure 17. Coefficient Input Section-Registers and Adders

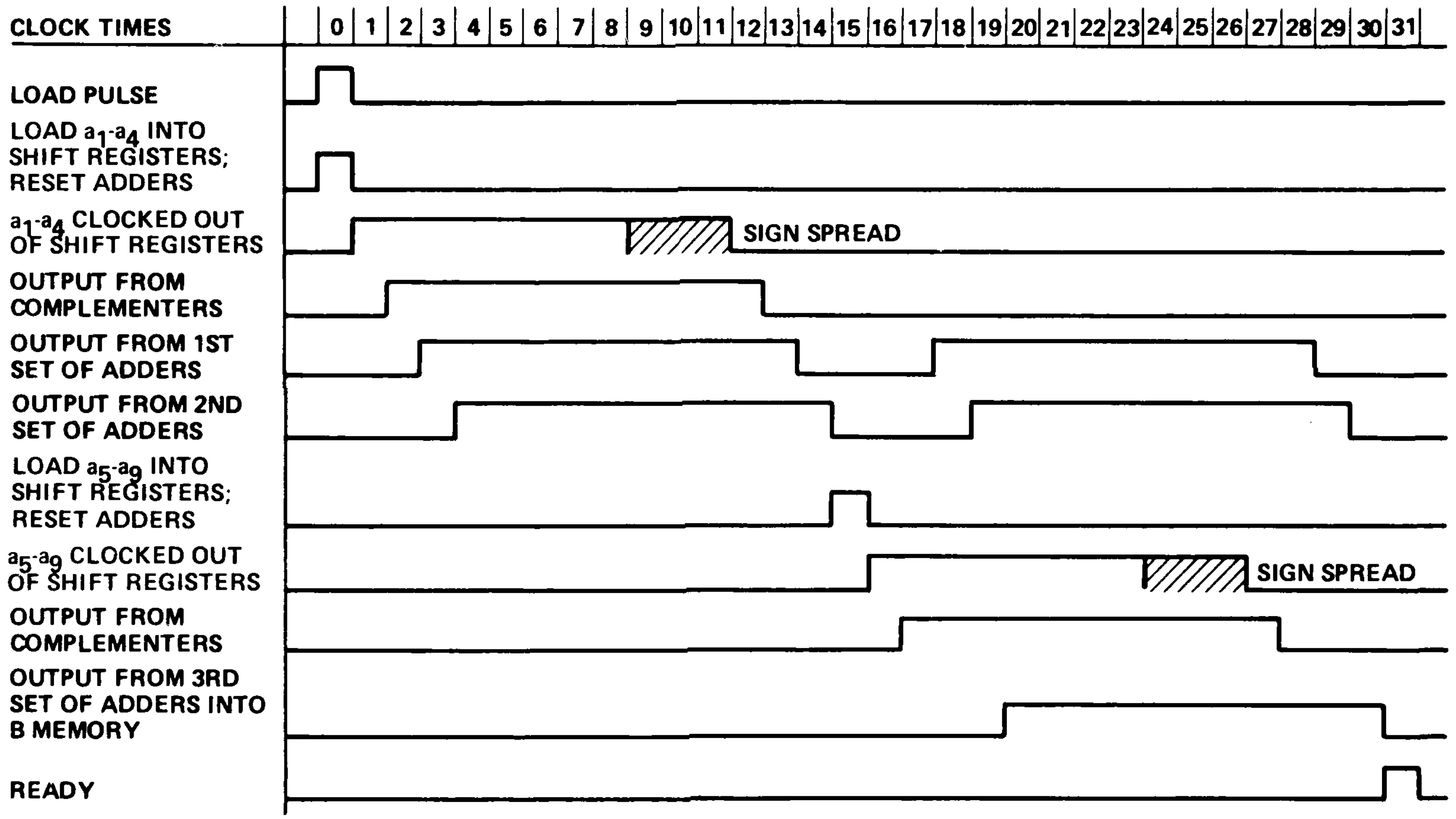

CLOCK TIMES
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
LOAD PULSE
LOAD a_1-a_4 INTO SHIFT REGISTERS; RESET ADDERS
a_1-a_4 CLOCKED OUT OF SHIFT REGISTERS
SIGN SPREAD
OUTPUT FROM COMPLEMENTERS
OUTPUT FROM 1ST SET OF ADDERS
OUTPUT FROM 2ND SET OF ADDERS
LOAD a_5-a_9 INTO SHIFT REGISTERS; RESET ADDERS
a_5-a_9 CLOCKED OUT OF SHIFT REGISTERS
SIGN SPREAD
OUTPUT FROM COMPLEMENTERS
OUTPUT FROM 3RD SET OF ADDERS INTO B MEMORY
READY

Figure 18. Coefficient Input Timing Diagram

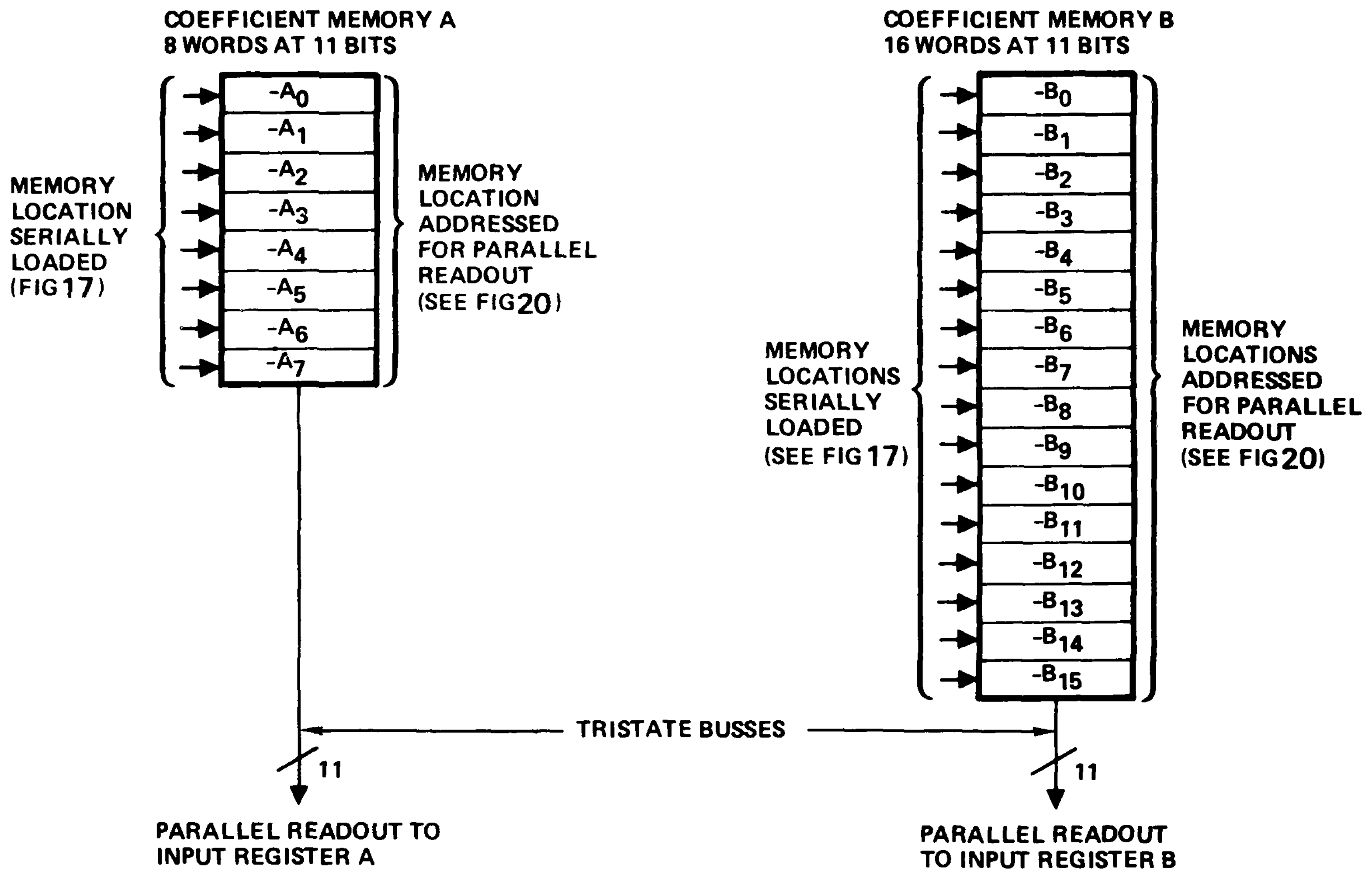
consists of an inverter followed by a single-input serial adder with a "1" preset in its carry flip flop. The 4 streams of serial data from the 4 complementers pass through the adder tree to generate $-A_0$ through $-A_7$. The last bit of the serial data is held at the output of each shift register for 3 additional clock periods in order to spread the sign and to drive the adder tree by 11-bit long data streams. This is a necessary step to ensure sign correction. By forming negative values rather than positive values, we can avoid some carry propagation problems later. Immediately after the formation of $-A_0$ through $-A_7$, registers a_5 through a_9 then parallel load shift registers 1 through 5 and the process which was described to generate $-A_0$ through $-A_7$, is repeated in order to generate $-B_0$ through $-B_{15}$, but using 5 data streams rather than 4.

While the PIPE is loading its coefficients as described above, the output from the arithmetic section is not valid, therefore, the output register (which is discussed in section 4.5.5) is forced to a RESET state. After the coefficient digestion has occurred, the coefficient load-control logic pauses until the output from the arithmetic section is again valid. At that time the output register is restored to normal operation.

4.5.2 Partial-Product-Memory Section

The organization of the memory section is shown in Figure 19. Eight serial data streams, $-A_0$ through $-A_7$, are serially clocked from the adder tree into partial-product memory A, thereby loading the upper half of Table 8 into the memory. Each of these values, $-A_0$ through $-A_7$, although loaded serially, will be read out in parallel onto a tristate bus which feeds the input register of arithmetic-section A (which is discussed below). The addressing means for reading out these coefficients is also discussed below.

Similarly, sixteen data streams, $-B_0$ through $-B_{15}$, are serially clocked from the adder tree into partial product memory B, thereby loading the upper half of Table 9 into the memory. Again, each of these values will be read out in parallel onto a tristate bus which feeds the input register of arithmetic section B.


COEFFICIENT MEMORY A
8 WORDS AT 11 BITS
MEMORY LOCATION SERIALLY LOADED (FIG 17)
$-A_0$
$-A_1$
$-A_2$
$-A_3$
$-A_4$
$-A_5$
$-A_6$
$-A_7$
MEMORY LOCATION ADDRESSED FOR PARALLEL READOUT (SEE FIG 20)
COEFFICIENT MEMORY B
16 WORDS AT 11 BITS
MEMORY LOCATIONS SERIALLY LOADED (SEE FIG 17)
$-B_0$
$-B_1$
$-B_2$
$-B_3$
$-B_4$
$-B_5$
$-B_6$
$-B_7$
$-B_8$
$-B_9$
$-B_{10}$
$-B_{11}$
$-B_{12}$
$-B_{13}$
$-B_{14}$
$-B_{15}$
MEMORY LOCATIONS ADDRESSED FOR PARALLEL READOUT (SEE FIG 20)
TRISTATE BUSSES
11
11
PARALLEL READOUT TO INPUT REGISTER A
PARALLEL READOUT TO INPUT REGISTER B

COEFFICIENT MEMORY

Figure 19. Memory Organization

The lower-half of Table 8 or 9 is effectively read into the arithmetic section by complementing its opposite-signed counterpart, e.g., $+A_0$ is entered by reading $-A_0$ from the memory, inverting each bit, and adding "1" through the carry input of the arithmetic section. This will be discussed later in greater detail.

4.5.3 Data Input Section

The data input section of Figure 20 performs three functions. The first function is the loading of the input data words into the appropriate shift registers, sometimes directly, and sometimes via the companion parallel register. In control state 0, no inputs are accepted. In control state 1, 9 parallel input words are accepted in succession. Switches SW1 and SW2 are closed and the load-mode-select logic provides a "read" signal to the parallel registers, 1 through 9, in succession. In control state 2 the switches SW1 and SW2 are open. The load-mode-select logic provides a "read" signal to parallel registers 1, 4 and 7 simultaneously; then to parallel registers 2, 5 and 8 simultaneously; then to parallel registers 3, 6, and 9 simultaneously. In control state 3 switches 1, 2, 5 and 8 are open, switches 3, 4, 6, 7, 9 and 10 are closed. After parallel loading registers 1, 4 and 7, the data are shifted. The data which are loaded into register 1 flows sequentially through shift registers 1, 2 and 3; the data which are loaded into register 4 flows sequentially through shift registers 4, 5 and 6; the data which are loaded into register 7 flows sequentially through shift registers 7, 8 and 9. In control state 4, switches 1 and 2 are open while all other switches are closed. After the input is parallel loaded into register 1, it is permitted to flow sequentially through all 9 shift registers. In control state 5, serial input data flow in through the SI port. Switches 1 and 2 remain open, switches 3 through 10 remain closed, and the data freely flow through shift registers 1 through 9 and out through the SO port.

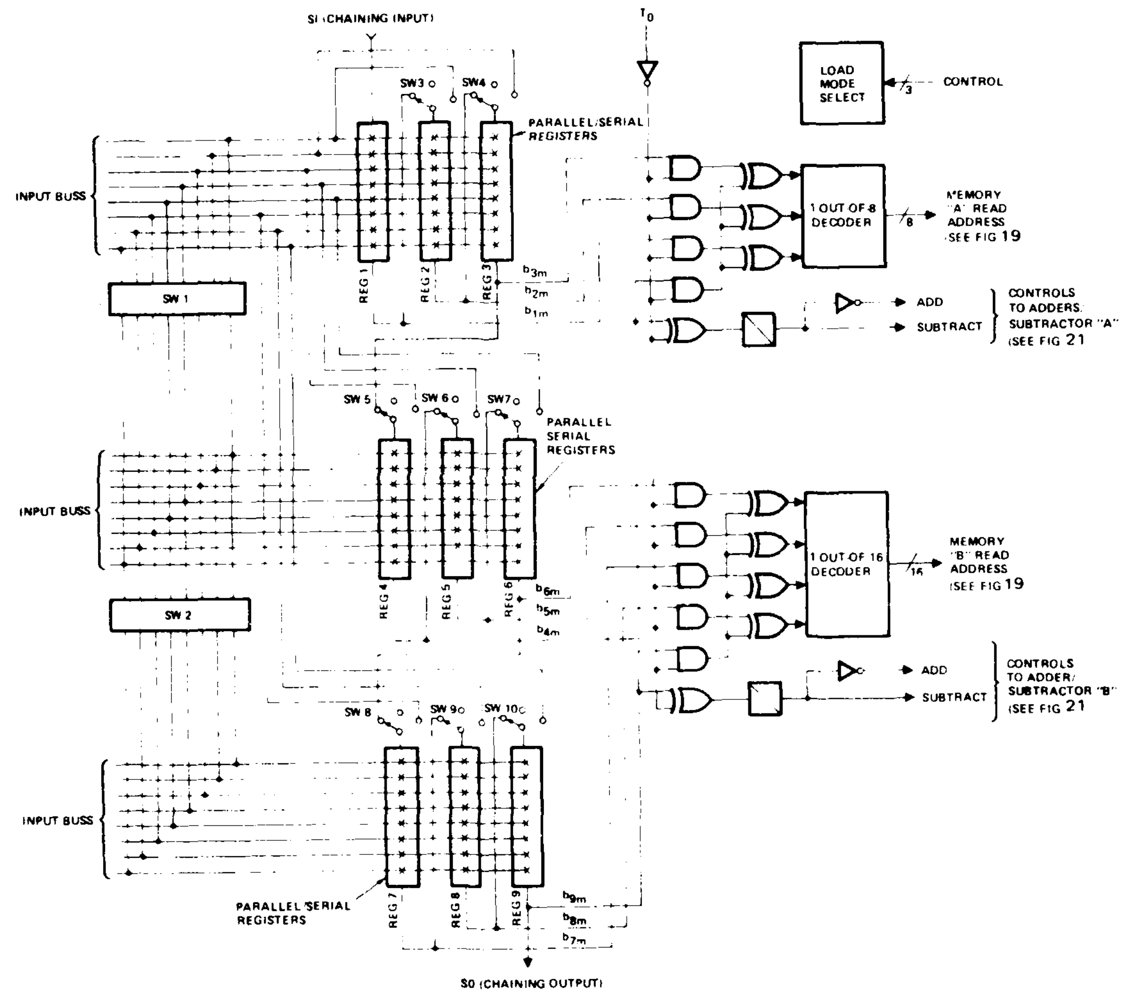


Figure 20. Data Input Section

In control state 6, the data are serially loaded into all 9 shift registers simultaneously. The input to the first shift register is through port SI. The second through ninth shift registers are loaded via the 8-bit bus and switches 3 through 10 as shown in Figure 20. In control state 7, three serial data words are loaded. The first data word is passed sequentially through the first three shift registers via input port SI and switches 3 and 4. The second word is passed sequentially through the second three shift register via switches 5, 6 and 7, and the third data word is similarly passed through the last three registers via switches 8, 9 and 10.

The purpose of the outputs of the registers is described in Tables 8 and 9. The outputs of the first 4 shift registers drive a 1-of-out-of 8 decoder. Forming the EXOR of b_{4m} with each of b_{1m}, b_{2m} and b_{3m} effectively folds the lower half of Table 8 onto the upper half; similarly the outputs of the last 5 shift registers drive a 1-out-of-16 decoder. Forming the EXOR of b_{9m} with each of b_{5m} b_{6m}, b_{7m}, and b_{8m} effectively folds the lower half of Table 9 onto the upper half. The outputs of the 4th and 9th shift registers also control the add/subtract functions of parallel adders, A and B respectively. By this means the adders select the appropriately signed inputs.

4.5.4 Arithmetic Section

Now we'll examine the arithmetic section which is composed of two adder/subtracter sections.

Refer to Figure 21. MUX1 of the "A" adder/subtracter accepts $-A_0$ through $-A_7$ which are the partial-product inputs from partial-product memory A, and selects the appropriately signed inputs. The 11-bit input is converted to a 12-bit input by sign spreading; i.e., the 11th and 12th bits are strapped together. MUX 2 selects either the initial condition from the initial-condition register or the previous result divided by 2. The LSB of the previous sum is gated into a carry generator, so no accuracy is lost. Similarly, MUX 1 of the "B" adder/subtracter accepts inputs $-B_0$ through $-B_{15}$, which are the inputs from partial-product memory B, and selects the appropriately signed inputs; MUX 2 of the "B" adder/subtracter selects the initial condition of the previous result divided by 2.

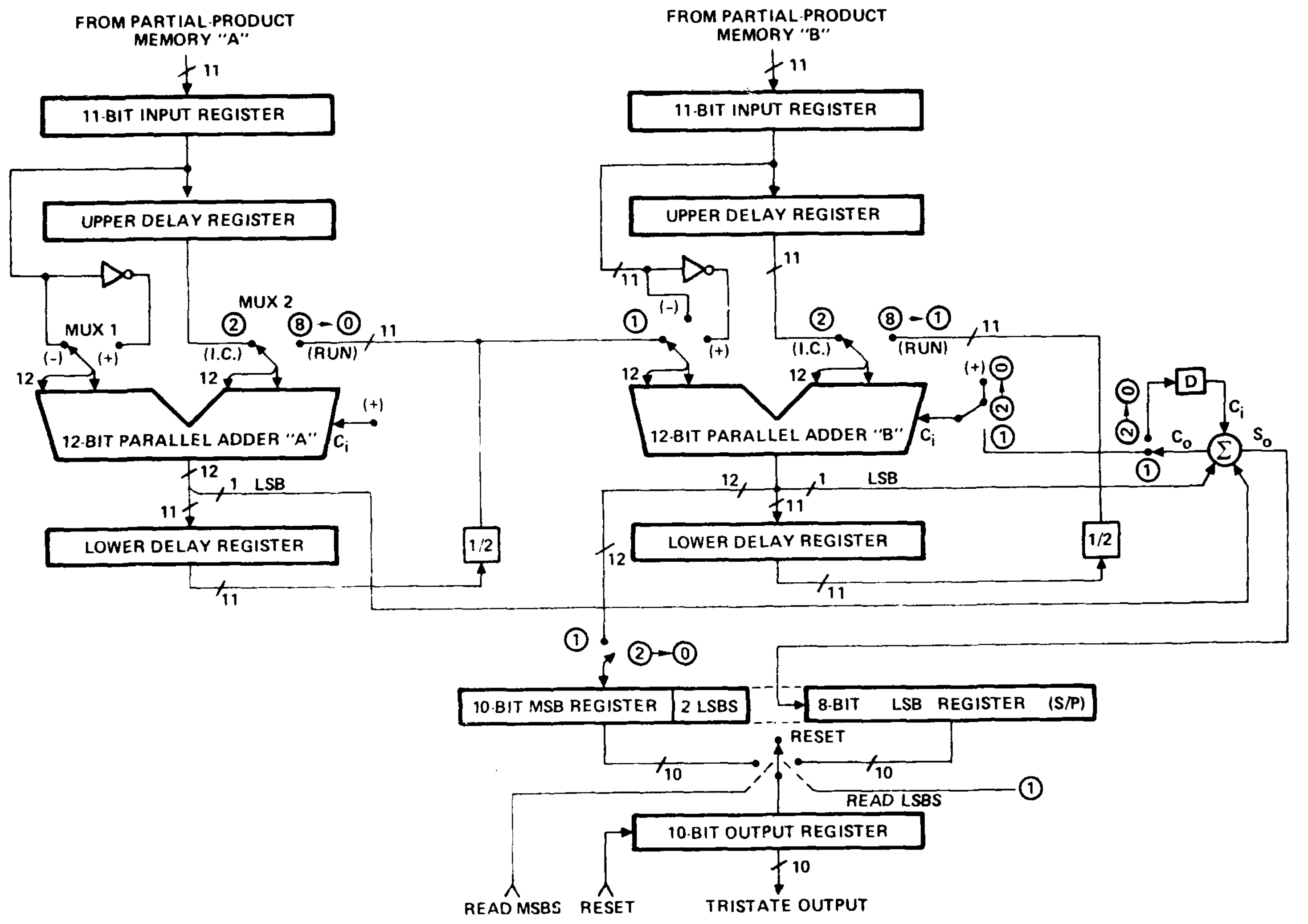


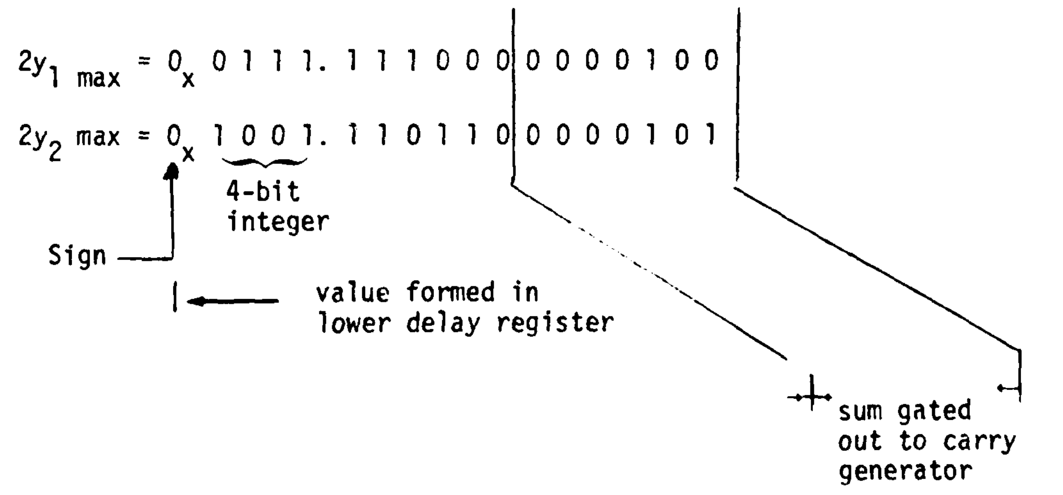
Figure 21. Pipe Arithmetic and Output Section

Since the values which are stored in the partial-product memories are $-A_0$ through $-A_7$ and $-B_0$ through $-B_{15}$, when the negative value is required, the memory content is simply gated directly through MUX 1. When the positive value is required, the memory content is inverted by complementing all bits, and a "1" is added through the carry input to the parallel adder.

After $2y_1$ and $2y_2$ are formed, the final result is obtained by gating $1/2(2y_1)$ through MUX 1 and $1/2(2y_2)$ through MUX 2 of the "B" adder/subtracter. The final carry is provided by the carry generator which sums the LSB's from both $2y_1$ and $2y_2$. The final most-significant 10-bit part of y is then gated into a holding register. The remaining 10 least-significant bits of the full-precision answer are also available. Lets see why we're interested in so many bits since we are permitting the word length of the input data word and of the coefficient to be each only 8 bits, including sign.

The product of a pair of 2's complement 8-bit numbers is 15 bits in length. The sum of 5 15-bit numbers is $14 + \log_2 5 = 18$ bits in length; the sum of 9 15-bit numbers is $15 + \log_2 9 = 19$ bits in length.

If we consider the input data words (the x_n) as fractions, their format will be the same as the format of the input coefficients. The value of y_1 can be as great as $2^2 \cdot (1-2^{-7})^2$; the value of y_2 can be as great as $(1+2^2) \cdot (1-2^7)^2$. Since the values which are actually formed by the adders are $2y_1$ and $2y_2$, their maximum values in binary format are



$2y_{1\,max} = 0_x\ 0\,1\,1\,1.\ 1\,1\,1\,0\,0\,0\ |\ 0\,0\,0\,0\,1\,0\,0$
$2y_2\ max = 0_x\ \underbrace{1\,0\,0\,1}.\ 1\,1\,0\,1\,1\,0\ |\ 0\,0\,0\,0\,1\,0\,1$
4-bit integer
Sign
value formed in lower delay register
sum gated out to carry generator

During the formation of $y = 1/2(2y_1 + 2y_2) \leq (1-2^{-7})^2$ we have the most significant 11 bits of y in the MSB register with a redundant sign bit for a total of 12 bits. The 2 LSB's are combined with the 8-bit carry-generator result. The answer is provided as a pair of 10-bit words, the 10 MSB's and the 10 LSB's.

The carry generator is a simple 3-input serial adder with the usual sum and carry outputs. During each of the 8 computational cycles each parallel adder sends the least-significant bit of its output to the carry generator. The carry generator in turn sends its "sum out" to an 8-bit shift register. The contents of the shift register are placed in the 8 LSB locations of the 10-bit output register. The other 2 bits which go into this register are the 2 LSB's of the MSB half of y. While the carry generator is providing its 8th "sum out" value, it is also providing a carry input to parallel adder B which is forming its final answer $y = 1/2 (2y_1 + 2y_2)$. A computing sequence is diagrammed in Figure 22.

4.5.5 Output Section

The output section which is also shown in Figure 21 consists of a 10-bit output register which drives a tristate bus, a 10-bit MSB register, a 10-bit LSB register, and the associated control and switching functions. The 10 LSB's are derived from two sources: the 8 LSB's are clocked from the carry generator into the 8-bit LSB serial/parallel register and held; this is augmented by another pair of bits which are the 2 LSB's from the final output of the adder B. This 10-bit result is then placed automatically into the output register. The 10 MSB's may be read into the output register by making the READ MSB line true. At any time, the output register may be cleared and the tristate bus allowed to float by making the RESET line true. The MSB's will not be read except upon command, but once every computational cycle the LSB's are read. Reading of the LSB's can be inhibited by holding the RESET line true.

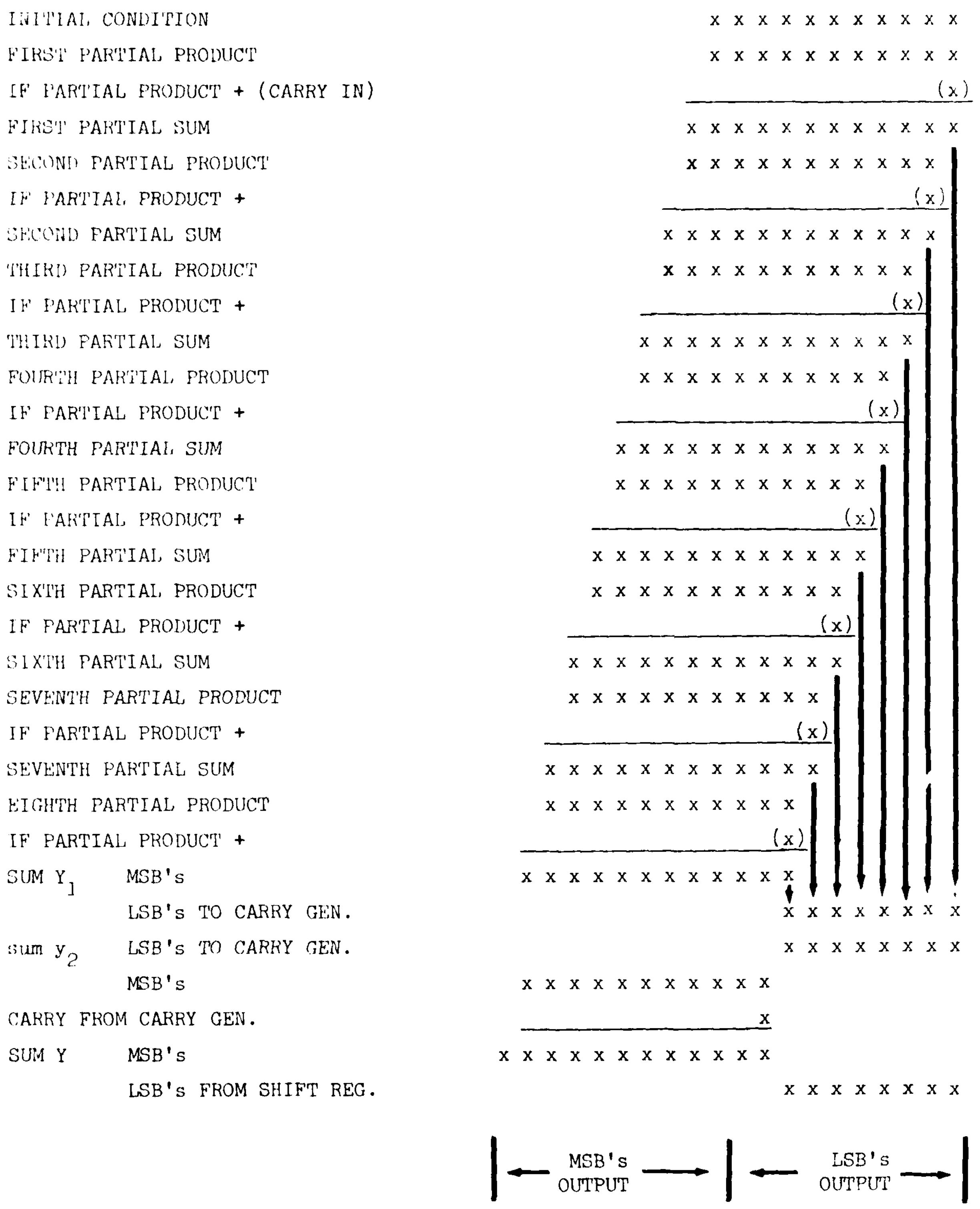

INITIAL CONDITION x x x x x x x x x x x
FIRST PARTIAL PRODUCT x x x x x x x x x x x
IF PARTIAL PRODUCT + (CARRY IN) (x)
FIRST PARTIAL SUM x x x x x x x x x x x x
SECOND PARTIAL PRODUCT x x x x x x x x x x x
IF PARTIAL PRODUCT + (x)
SECOND PARTIAL SUM x x x x x x x x x x x x
THIRD PARTIAL PRODUCT x x x x x x x x x x x
IF PARTIAL PRODUCT + (x)
THIRD PARTIAL SUM x x x x x x x x x x x x
FOURTH PARTIAL PRODUCT x x x x x x x x x x x
IF PARTIAL PRODUCT + (x)
FOURTH PARTIAL SUM x x x x x x x x x x x x
FIFTH PARTIAL PRODUCT x x x x x x x x x x x
IF PARTIAL PRODUCT + (x)
FIFTH PARTIAL SUM x x x x x x x x x x x x
SIXTH PARTIAL PRODUCT x x x x x x x x x x x
IF PARTIAL PRODUCT + (x)
SIXTH PARTIAL SUM x x x x x x x x x x x x
SEVENTH PARTIAL PRODUCT x x x x x x x x x x x
IF PARTIAL PRODUCT + (x)
SEVENTH PARTIAL SUM x x x x x x x x x x x x
EIGHTH PARTIAL PRODUCT x x x x x x x x x x x
IF PARTIAL PRODUCT + (x)
SUM Y1 MSB's x x x x x x x x x x x x
LSB's TO CARRY GEN. x x x x x x x x
sum y2 LSB's TO CARRY GEN. x x x x x x x x
MSB's x x x x x x x x x x x
CARRY FROM CARRY GEN. x
SUM Y MSB's x x x x x x x x x x x x
LSB's FROM SHIFT REG. x x x x x x x x
MSB's OUTPUT
LSB's OUTPUT

Figure 22. Computing Sequence

A forced RESET occurs when a new set of coefficients is being digested. This RESET will be held until the output of the arithmetic section is again valid.

4.5.6 Timing and Control Section

Figure 23 shows the timing of all the arithmetic operations. The timing requirements are extremely simple. Simple delay, invert, and NAND operations on the input-word-timing pulse can provide all necessary internal timing signals for the arithmetic section as shown in Figure 24.

4.5.7 Overall Chip Structure

Figure 25 illustrates the entire chip and the relationship between the various parts which were described above. The complete set of logic diagrams which were developed is in the Appendix.

4.5.8 Using Multiple Devices for High Accuracy

The high-accuracy computation mode can be understood by considering a set of 15-bit inputs which can be expressed as the following

$$x_n = -b_{no} + \sum_{m=1}^{14} b_{nm} 2^{-m} = [-b_{no} + \sum_{m=1}^{7} b_{nm} 2^{-m}] + [0 + \sum_{m=1}^{7} b_{nm+7} 2^{-m}] 2^{-7}$$

$$= x_{Mn} + x_{Ln} 2^{-7}$$

where x_{Mn} are the most significant bits of x_n and x_{Ln} are the least significant bits. We can similarly express the coefficients $a_n = a_{Mn} + a_{Ln}2^{-7}$ so the sum of the products can be written as

$$y = \sum_{n=1}^{9} a_n x_n = \sum_{n=1}^{9} (a_{Mn} + a_{Ln} 2^{-7})(x_{Mn} + x_{Ln} 2^{-7})$$

$$= \sum_{n=1}^{9} a_{Mn} x_{Mn} + [\sum_{n=1}^{9} a_{Ln} x_{Mn} + \sum_{n=1}^{9} a_{Mn} x_{Ln}] 2^{-7} + [\sum_{n=1}^{9} a_{Ln} x_{Ln}] 2^{-14}$$

In order to perform multiplication with an input accuracy of 15 bits and an output accuracy of 33 bits requires a set of four PIPE devices plus three external adders.

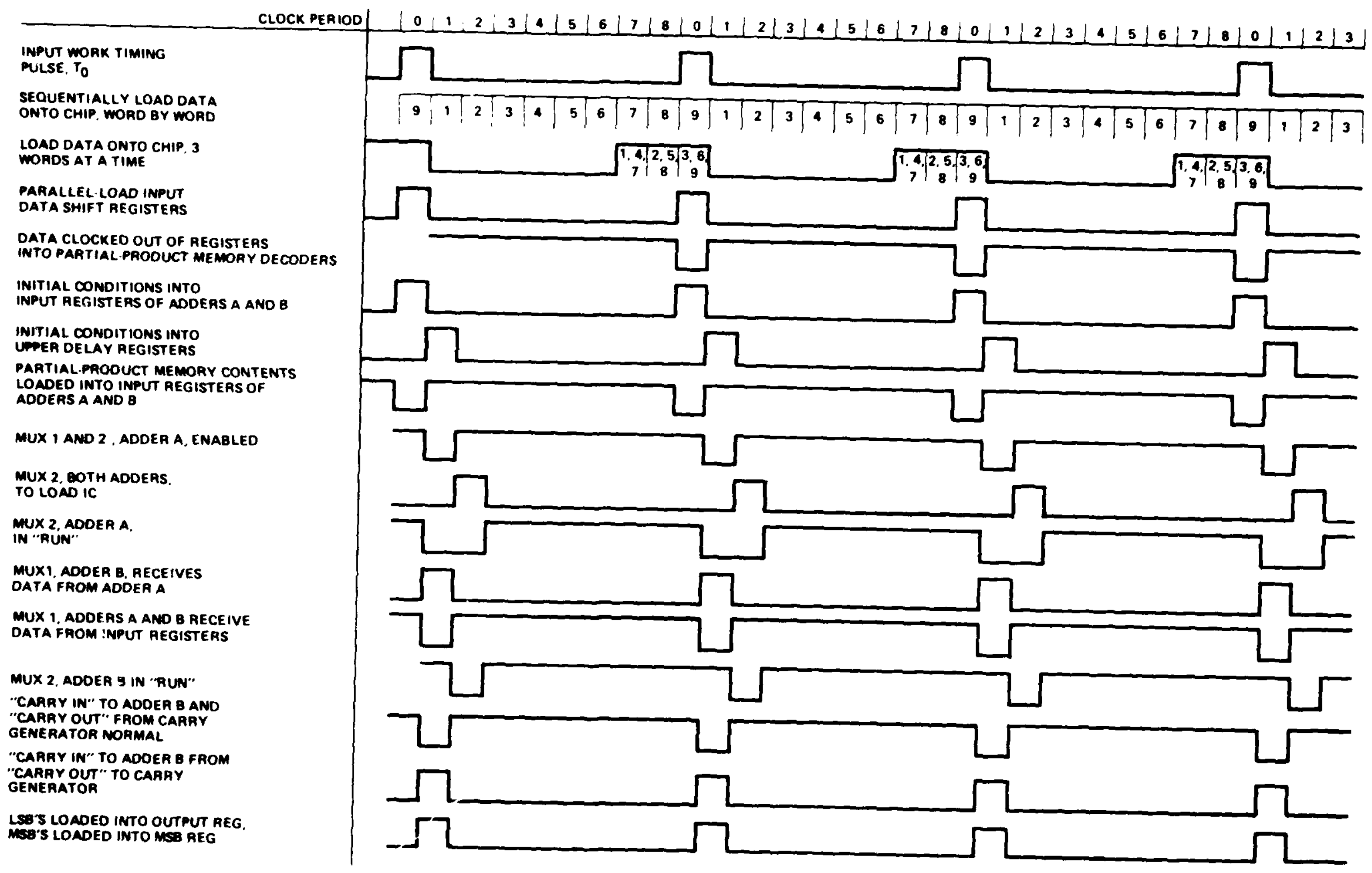

CLOCK PERIOD
0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3
INPUT WORK TIMING PULSE, T_0
SEQUENTIALLY LOAD DATA ONTO CHIP, WORD BY WORD
9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9 1 2 3
LOAD DATA ONTO CHIP, 3 WORDS AT A TIME
1, 4, 7
2, 5, 8
3, 6, 9
PARALLEL-LOAD INPUT DATA SHIFT REGISTERS
DATA CLOCKED OUT OF REGISTERS INTO PARTIAL-PRODUCT MEMORY DECODERS
INITIAL CONDITIONS INTO INPUT REGISTERS OF ADDERS A AND B
INITIAL CONDITIONS INTO UPPER DELAY REGISTERS
PARTIAL-PRODUCT MEMORY CONTENTS LOADED INTO INPUT REGISTERS OF ADDERS A AND B
MUX 1 AND 2 , ADDER A, ENABLED
MUX 2, BOTH ADDERS, TO LOAD IC
MUX 2, ADDER A, IN "RUN"
MUX1, ADDER B, RECEIVES DATA FROM ADDER A
MUX 1, ADDERS A AND B RECEIVE DATA FROM INPUT REGISTERS
MUX 2, ADDER B IN "RUN"
"CARRY IN" TO ADDER B AND "CARRY OUT" FROM CARRY GENERATOR NORMAL
"CARRY IN" TO ADDER B FROM "CARRY OUT" TO CARRY GENERATOR
LSB'S LOADED INTO OUTPUT REG, MSB'S LOADED INTO MSB REG

Figure 23. Timing Diagram for Arithmetic Section

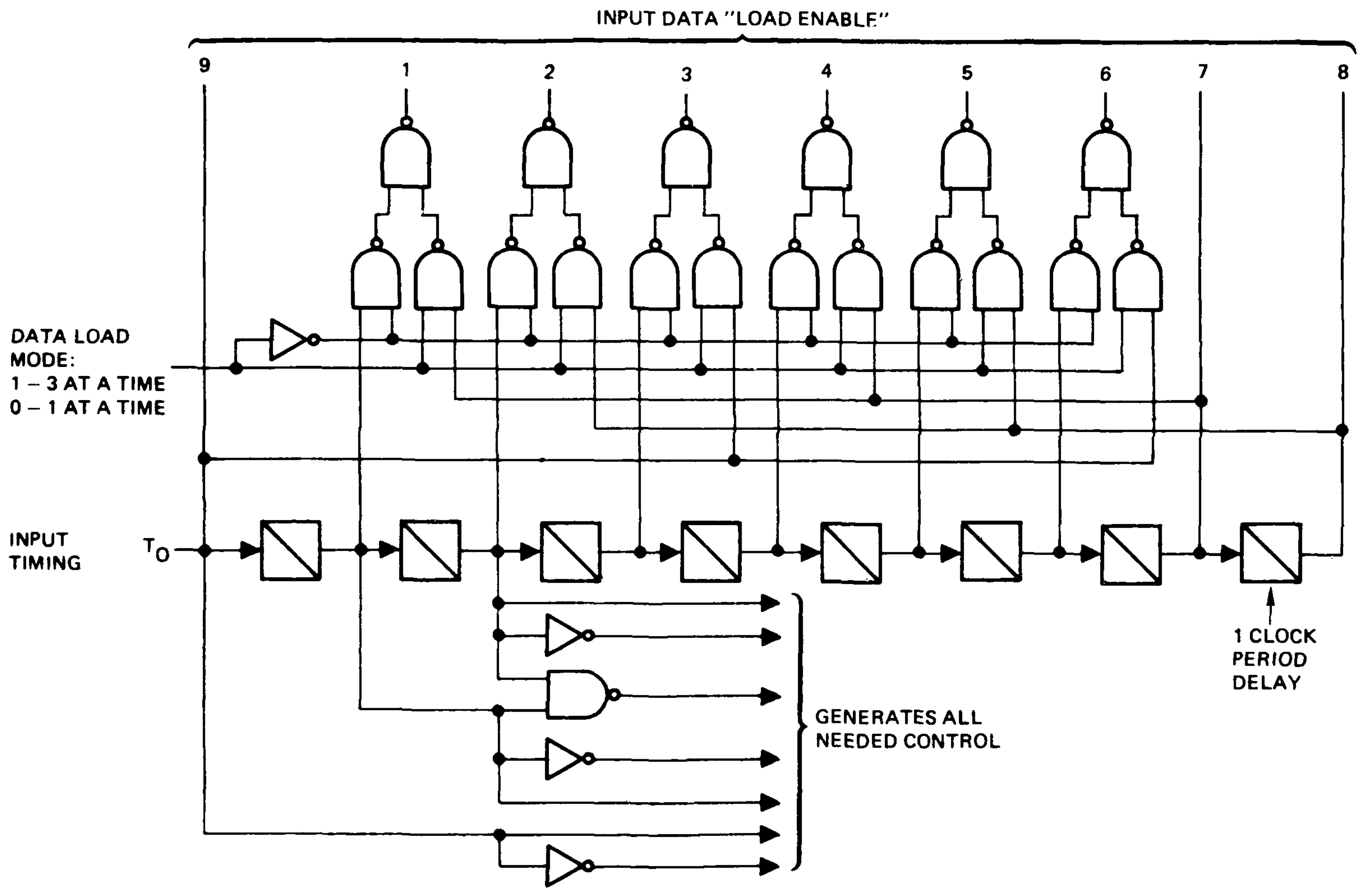


Figure 24. Arithmetic and Data Timing and Control Generator

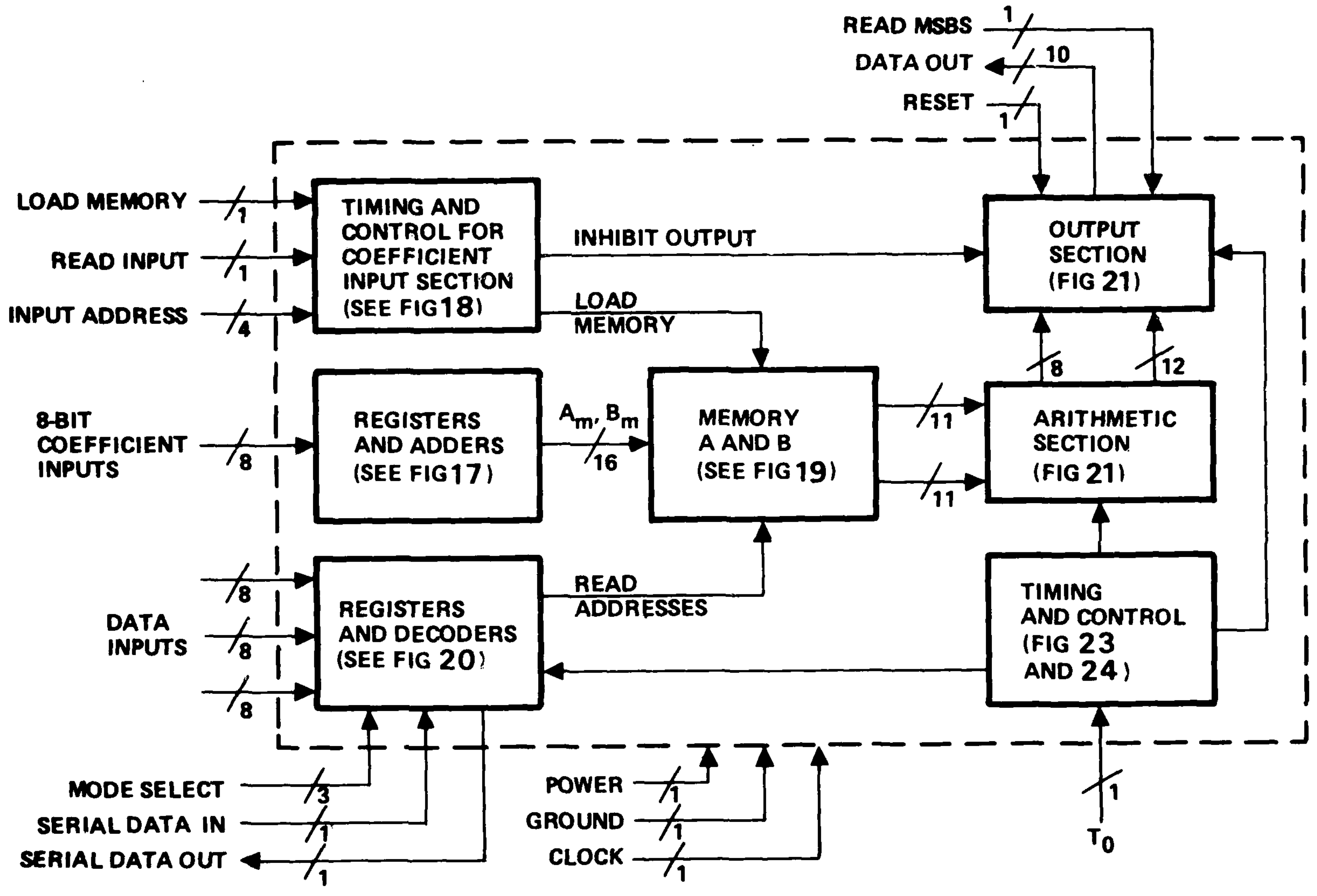


Figure 25. Pipe Chip Organization

APPENDIX

LOGIC DESIGN AND LAYOUT

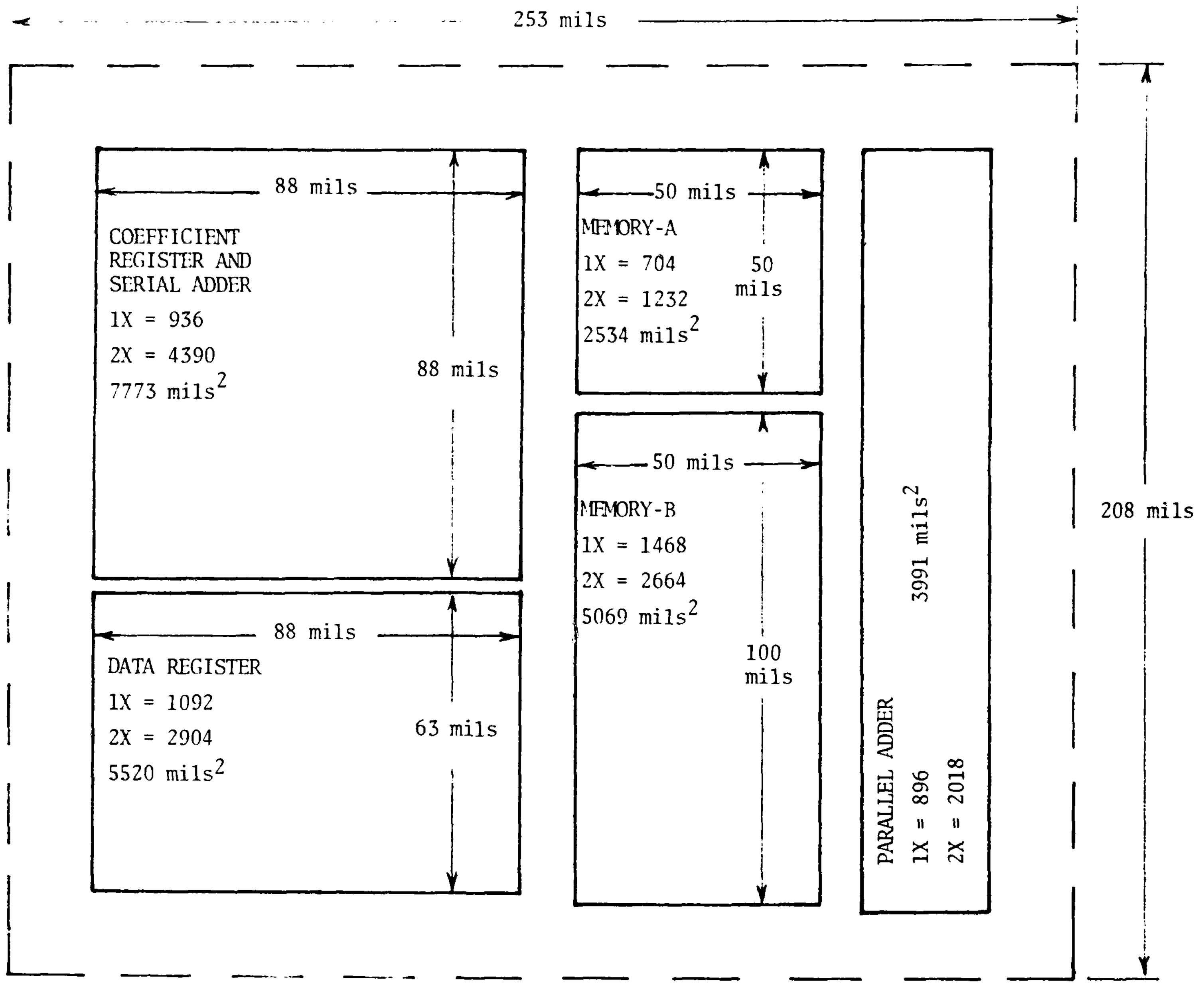


Figure 26. PIPE Integrated Circuit Partitioning

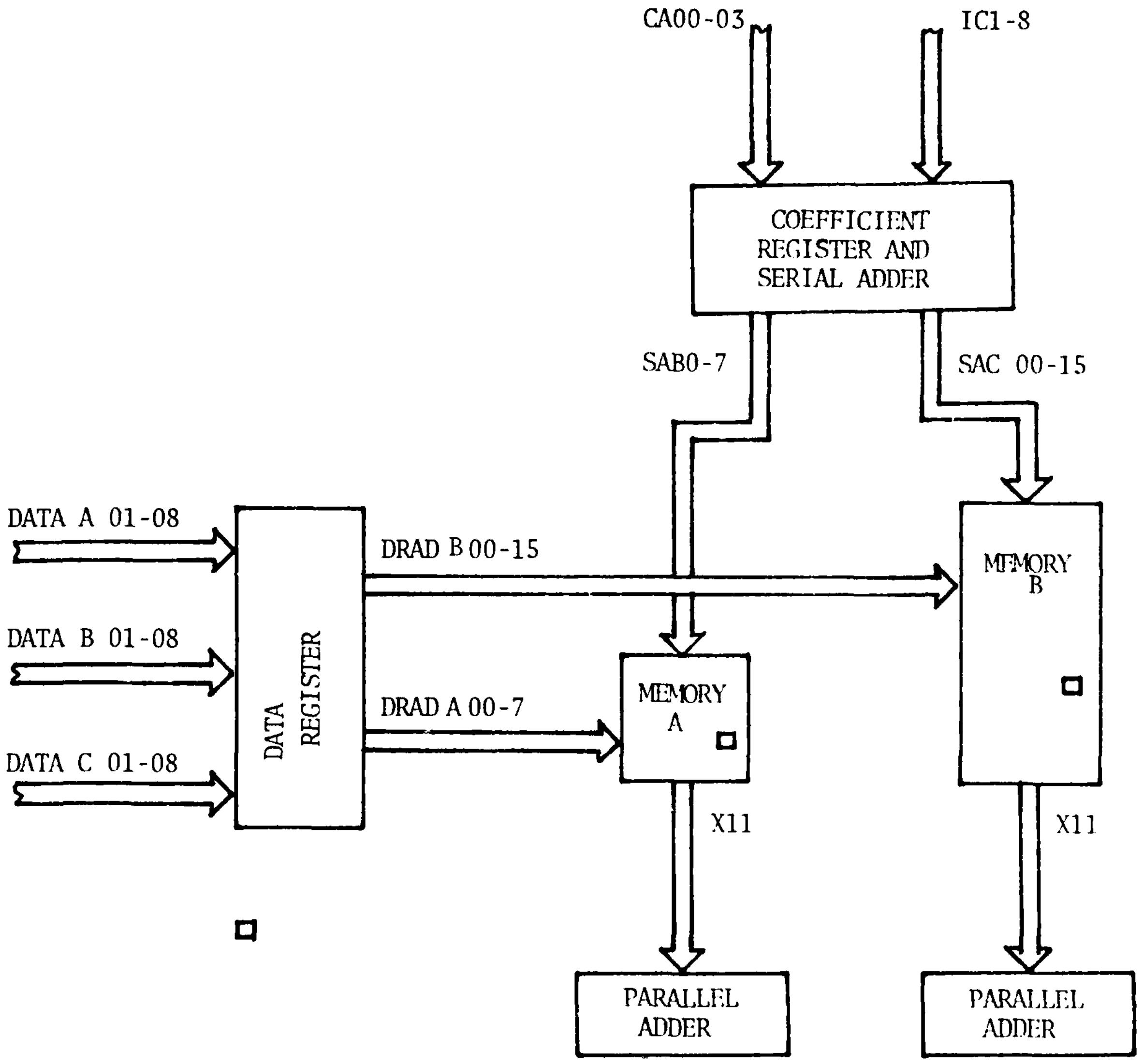


Figure 27. PIPE Integrated Circuit Block Diagram

REFERENCES

1. J.M.S. Prewitt, "Object Enhancement and Extraction," Picture Processing and Psychopictorics, B. S. Lipkin and A. Rosenfield, editors; Academic Press, New York, 1970, pg. 126.

2. G. S. Robinson, "Detection and Coding of Edges Using Directional Masks," SPIE Vol. 87, Advances in Image Transmission Techniques, 1976, pp. 117-125.

3. A. Peled, B. Liu, Digital Signal Processing, John Wiley & Sons, New York, 1976, Chapt. 5.

4. S. A. White, On Mechanization of Vector Multiplication, Proc. IEEE, Vol. 63, pp. 730-731, April 1975.

5. M. Buttner and H. W. Schussler, "On Structures for the Implementation of Distributed Arithmetic, NTZ Communications Journal, Vol. 6, June 1975.

6. S. A. White, "An Adaptive Recursive Digital Filter," Proc. 9th Asilomar Conference on Circuits, Systems, and Computers, Pacific Grove, CA, Nov. 1975.

7. T. L. Chang, "A Low Roundoff Noise Digital Filter Structure," Proc. IEEE ISCAS, May 1978.

8. A. D. Booth, "A Signed Binary Multiplication Technique," Quarterly Journal of Mechanics and Applied Mathematics, Vol. 4, part 2, 1951.

9. O. L. MacSorley, "High-Speed Arithmetic in Binary Computers," Proc. IRE, Jan. 1961, pp. 67-91.

☆U.S.Government Printing Office: 1981 — 757-002/572

DATE
FILMED

- 8